Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

ИНФОРМАТИКА

Пособие для подготовки к вступительным экзаменам, проводимым ТУСУРом самостоятельно по дисциплине «ИНФОРМАТИКА»

2018

Потахова Ирина Владимировна

Пособие для подготовки к вступительным экзаменам, проводимым ТУСУРом самостоятельно по дисциплине «ИНФОРМАТИКА / И.В. Потахова. – Томск, 2018. – 22 с.

Оглавление (содержание)

1. ВВЕДЕНИЕ…… 8

2. КОМПЬЮТЕР И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ…… 11

2.1. Процессор и память…… 11

2.1.1. Процессор…… 11

2.1.2. Оперативная память…… 12

2.1.3. Внешняя память…… 13

2.2. Основные компоненты персонального компьютера…… 15

2.2.1. Системный блок…… 15

2.2.2. Периферия…… 16

2.2.2.1. Клавиатура…… 16

2.2.2.2. Мышь…… 17

2.2.2.3. Монитор…… 17

2.2.2.4. Сканер, принтер…… 18

2.2.3. Задания…… 19

2.3. Программы…… 20

2.3.1. Интерфейс программы, его виды…… 20

2.3.2. Операционная система, ее компоненты…… 21

2.3.3. Виды операционных систем…… 22

2.3.4. Утилиты…… 23

2.3.5. Прикладные программы…… 24

2.3.5.1. Текстовые редакторы…… 25

2.3.5.2. Графические и мультимедийные редакторы…… 26

2.3.5.3. Редакторы баз данных…… 26

2.3.5.4. Языки программирования…… 26

2.4. Файл и дерево каталогов…… 27

2.4.1. Имена дисков…… 28

2.4.2. Каталог…… 29

2.4.3. Дерево каталогов, надкаталог…… 29

2.4.4. Создание дерева каталогов, подкаталог…… 29

2.4.5. Добавление новой информации…… 30
2.4.6. Имена и содержание каталогов…… 30
2.4.7. Задания……... 31
3. ИНФОРМАЦИЯ, КОДИРОВАНИЕ ИНФОРМАЦИИ……. 32
3.1. Понятие «информация» и свойства информации…….. 32
3.2. Количество информации…… 34
3.2.1. Вопросы и задания…….. 38
3.3. Алфавитный подход к определению количества информации 38
3.4. Представление и кодирование информации……... 39
3.4.1. Язык как знаковая система…… 39
3.4.2. Кодирование информации…….. 41
3.4.3. Представление информации в компьютере…… 41
3.5. Системы счисления…… 45
3.5.1.Основные понятия систем счисления……..... 46
3.5.2. Правила перевода чисел из одной системы счисления в другую 52
3.5.3. Арифметические действия над целыми числами в 2-ой системе счисления…… 57
3.5.4. Сложение и вычитание в восьмеричной системе счисления… 58
3.5.5. Сложение и вычитание в шестнадцатеричной системе счисления 58
3.5.6. Контрольные вопросы…… 59
3.5.7. Задания…….. 61
4. ОСНОВЫ ЛОГИКИ…… 62
4.1. Формы мышления…… 62
4.1.1. Вопросы…… 66
4.2. Алгебра высказываний……... 66
4.2.1. Логическое умножение (конъюнкция)…… 67
4.2.2. Логическое сложение (дизъюнкция)…… 68

4.2.3. Логическое отрицание (инверсия)…… 70
4.2.4. Логические выражения и таблицы истинности…… 71
4.2.4.1. Задания…… 74
4.2.5. Логические функции…… 74
4.2.5.1. Вопросы и задания…… 78
4.3 Логические законы и правила преобразования логических выражений…… 78
4.3.1. Задания…… 80
4.4. Решение логических задач…… 80
4.4.1. Задания…… 85
5. ПРОГРАММИРОВАНИЕ…… 90
5.1. Алгоритм, его характеристики…… 90
5.2. Создание алгоритмов…… 92
5.3. Модульность алгоритмов, головная программа…… 92
5.4. Технология программирования…… 94
5.4.1. Структурное программирование…… 94
5.4.2. Объектно–ориентированное программирование…… 95
5.4.3. Визуальное программирование…… 95
5.5. Кодирование и исполнение программ…… 96
5.6. Ошибки программирования…… 97
5.7. Этапы создания программ…… 97
5.8. Описание алгоритма…… 98
5.9. Элементарные структурные алгоритмы…… 100
5.10. Контрольные вопросы…… 106
5.11. Задания…… 107
6. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ…… 108
6.1. Немного истории…… 108
6.2. Алфавит языка…… 109
6.3. Структура программы…… 109
6.4. Данные в Паскале…… 111

6.4.1. Типы данных..... 112
6.4.2. Константы 114
6.4.3. Переменные элементарного типа..... 115
6.4.4. Контрольные вопросы..... 116
6.5. Преобразование данных..... 116
6.5.1. Операции. Выражения. Правила построения выражений.... 116
6.5.2. Функции обработки числовых данных..... 118
6.5.3. Задания..... 120
6.6. Операторы..... 121
6.6.1. Простые операторы..... 121
6.6.1.1. Оператор комментария..... 121
6.6.1.2. Оператор присваивания..... 122
6.6.1.3. Операторы процедуры..... 124
6.6.1.4. Контрольные вопросы и задания..... 129
6.6.2. Сложные операторы..... 130
6.6.2.1. Составной оператор..... 130
6.6.2.2. Управляющие операторы..... 131
6.6.2.2.1. Контрольные вопросы..... 135
6.6.2.2.2. Задания..... 136
6.6.2.3. Цикл с параметром (For…)..... 137
6.6.2.3.1. Задания..... 138
6.6.2.4. Циклы с условием 140
6.6.2.4.1. Задания..... 144
6.6.2.4.2. Контрольные вопросы..... 144
6.7. Массивы..... 145
6.8. Базовые алгоритмы на массивах..... 148
6.8.1. Вычисление суммы элементов числовых массивов..... 148
6.8.2. Вычисление произведения элементов числовых массивов. 149
6.8.3. Поиск минимального и максимального значений в массиве 150
6.8.4. Задания..... 152

6.9. Сортировка массивов..... 153
6.9.1. Простые методы сортировки..... 155
6.9.2. Усовершенствованные методы сортировки..... 160
6.9.3. Задания..... 165
6.10. Поиск заданного элемента в массиве..... 166
6.10.1. Вопросы и задания..... 168
6.11. Строки..... 169
6.11.1. Задания..... 172
6.12. Записи..... 173
6.12.1. Задания..... 176
6.13. Процедуры и функции 178
6.13.1. Синтаксис описания процедур и функций..... 184
6.13.2. Параметры процедур и функций..... 185
6.13.3. Вызов процедур и функций..... 187
6.13.4. Параметры–массивы..... 189
6.13.5. Контрольные вопросы..... 191
6.13.7. Задания..... 191
Литература..... 232

1. ВВЕДЕНИЕ

Компьютерные науки и информационные технологии стали вездесущими и продолжают сулить перемены, которые еще больше затрагивают практически все сферы нашей жизни. Компьютеры превратились в неотъемлемую часть современной культуры и являются движущей силой экономического роста во всем мире. Информатика развивается с поразительной скоростью. Постоянно появляются новые технологии, а существующие становятся устаревшими. В такой ситуации выигрывают люди, которые могут учиться, умеют учиться и обладают необходимыми базовыми знаниями.

Информатика — (от французского *information* — информация и automatioque — автоматика) — это область научно-технической деятельности, занимающаяся исследованием процессов получения, передачи, обработки, хранения и представления информации, решением проблем создания, внедрения и использования информационной техники и технологии во всех сферах общественной жизни.

Основная задача информатики заключается в определении общих закономерностей, в соответствии с которыми происходит создание научной информации, ее преобразование, передача и использование в различных сферах деятельности человека. Прикладные задачи заключаются в разработке более эффективных методов и средств осуществления информационных процессов, в определении способов оптимальной научной коммуникации с широким применением технических средств.

Современный подход к ее изучению информатики рассматривает объединение целого ряда самостоятельных областей знаний, каждая из которых имеет свою специфику. К ним относятся дискретные структуры, основы программирования, алгоритмы и теории сложности, архитектура и организация ЭВМ, операционные системы, социальные и профессиональные вопросы программирования.

В структуре информатики как науки выделяют алгоритмическую, программную техническую области. Информатика входит в состав кибернетики, изучающей общую теорию управления и передачи информации. Кибернетика — наука об общих законах получения, хранения, передачи и обработки информации в сложных системах. Под сложными системами понимаются технические, биологические и социальные системы. Кибернетика пригодна для исследования любой системы, которая может записывать, накапливать и обрабатывать информацию, благодаря чему ее можно использовать в целях управления.

В настоящее время информатика вполне сформировалась как академическая дисциплина и находится в периоде бурного развития. Она властно вторгается в производство, бизнес, медицину и многие другие области.

Расширение предметной области применения информатики превращает ее из технической дисциплины о методах и средствах обработки данных при помощи средств вычислительной техники в фундаментальную естественную науку об информации и информационных процессах в природе и обществе.

На сегодняшний день информатика является не просто важной дисциплиной, она также обслуживает множество различных других наук. Муниципальный чиновник, работающий с демографической базой данных, экономист, создающий компьютерные модели — это все примеры людей, применяющих информационные технологии в своей профессиональной деятельности. Трудно определить информатику как единую дисциплину. Современный подход к ее изучению рассматривает объединение целого ряда самостоятельных областей знаний, каждая из которых имеет свою специфику. К ним относятся дискретные структуры, основы программирования, алгоритмы и теории сложности, архитектура и организация ЭВМ, операционные системы, социальные и профессиональные вопросы программирования. Наша учеба, работа, личные дела — это ка-

ждодневное, ежечасное решение различных задач. Каждая задача требует для своего решения выполнения определенных действий. Многократно решая задачи, можно заметить, что необходимые действия должны выполняться в строго определенном порядке. В таких случаях принято говорить об алгоритме решения задач и программах.

Знание основ программирования является необходимым условием для освоения большинства разделов информатики.

Цель преподавания дисциплины «Информатика» состоит в изучении основных положений и разделов информатики; получении навыков практического использования компьютера; получении отчетливого представления о роли информатики и информационных технологий в современном мире.

Задачами изучения дисциплины являются:

- развитие логического и алгоритмического мышления;
- овладение основами функционирования персональных компьютеров, методами и средствами хранения, обработки и передачи информации;
- выработка умения самостоятельного решения алгоритмических задач обработки текстовой и цифровой информации, навыков практической работы на персональном компьютере.

Настоящее пособие представляет собой руководство по информатике для учащихся 11 класса физико–математического лицея ТУСУРа, готовящихся к поступлению в Томский государственный университет государственный университет систем управления и радиоэлектроники.

2. КОМПЬЮТЕР И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

2.1. Процессор и память

2.1.1. Процессор

***Процессор**, или ЦПУ (CPU)* — это «мозг» компьютера, физически представляет собой большую интегральную микросхему, полупроводниковый кристалл. Процессор выполняет арифметические операции с двоичными числами.

К основным характеристикам процессора относятся:

– Быстродействие (вычислительная мощность) – это среднее число операций процессора в секунду.

– Тактовая частота в МГц. Тактовая частота равна количеству тактов в секунду. Такт — это промежуток времени между началом подачи текущего импульса ГТЧ и началом подачи следующего. Характерные тактовые частоты микропроцессоров: 40 МГц, 66 МГц, 100 МГц, 130 МГц, 166 МГц, 200 МГц, 333 МГц, 400 МГц, 600 МГц, 800 МГц, 1000 МГц и т. д. До 3ГГц Тактовая частота отражает уровень промышленной технологии, по которой изготавливался данный процессор. Она также характеризирует и компьютер, поэтому по названию модели микропроцессора можно составить достаточно полное представление о том, к какому классу принадлежит компьютер. Поэтому часто компьютерам дают имена микропроцессоров, входящих в их состав. Ниже приведены названия наиболее массовых процессоров, выпущенных фирмой *Intel* и годы их создания: 8080 (1974 г.), 80286 (1982 г.), 80386*DX* (1985 г.), 80486*DX* (1989 г.), 80586 или *Pentium* (1993 г.), *Pentium Pro* (1995 г.), *Pentium* II (1997 г.), *Pentium III* (1999 г.), *Pentium IV* (2001 г.). Как видно, увеличение частоты – одна из основных тенденций развития микропроцессоров. На рынке массовых компьютеров лидирующее место среди производителей процессоров занимают 2 фирмы: *Intel* и *AMD*. За ними закрепилось базовое название, переходящее от модели к модели. У *Intel* —это

Pentium и модель с урезанной кэш–памятью *Pentium Celeron*; у *AMD* — это *Athlon* и модель с урезанной кэш–памятью *Duron*.

– Разрядность процессора — это максимальное количество бит информации, которые могут обрабатываться и передаваться процессором одновременно. Разрядность процессора определяется разрядностью регистров, в которые помещаются обрабатываемые данные. Например, если регистр имеет разрядность 2 байта, то разрядность процессора равна 16 (2x8); если 4 байта, то 32; если 8 байтов, то 64.

Для пользователей процессор интересен, прежде всего, своей системой команд и скоростью их выполнения. Система команд процессора представляет собой набор отдельных операций, которые может выполнить процессор данного типа. Разные модели микропроцессоров выполняют одни и те же операции за разное число тактов. Чем выше модель микропроцессора, тем, как правило, меньше тактов требуется для выполнения одних и тех же операций.

Для математических вычислений к основному микропроцессору добавляют математический сопроцессор. Начиная с модели 80486*DX*, процессор и сопроцессор выполняют на одном кристалле.

2.1.2. Оперативная память

Памятью компьютера называется совокупность устройств для хранения программ, вводимой информации, промежуточных результатов и выходных данных. ***Оперативная,*** или временная, память функционально и конструктивно связана с процессором. В оперативной памяти в двоичном виде запоминается обрабатываемая информация, программа ее обработки, промежуточные данные и результаты работы. Оперативная память обеспечивает режимы записи, считывания и хранения информации, причём в любой момент времени возможен доступ к любой произвольно выбранной ячейке памяти.

Однако информация, которую компьютер записывает во временную память, исчезает при его выключении.

2.1.3. Внешняя память

Для постоянного хранения информации используется *внешняя память* компьютера, находящаяся в системном блоке в виде отдельного устройства. Это устройство называется *жесткий диск, или винчестер (hard disk drive, HDD).* Информация, записанная на магнитной поверхности винчестера, хранится и после выключения компьютера.

Выделяют следующие основные типы устройств памяти с произвольным доступом:

1. Накопители на жёстких магнитных дисках (винчестеры, НЖМД) — несъемные жесткие магнитные диски. Ёмкость современных винчестеров от сотен мегабайт до нескольких сотен гигабайт. На современных компьютерах это основной вид внешней памяти. Первые жесткие диски состояли из 2 дисков по 30 Мбайт и обозначались 30/30, что совпадало с маркировкой модели охотничьего ружья “Винчестер” — отсюда пошло такое название этих накопителей.
2. Накопители на гибких магнитных дисках (флоппи–дисководы, НГМД) – устройства для записи и считывания информации с небольших съемных магнитных дисков (дискет), упакованных в пластиковый конверт. Максимальная ёмкость дискеты — 1,44 Мбайт.
3. Оптические диски (CD–ROM — Compact Disk Read Only Memory) — компьютерные устройства для чтения с компакт-дисков. Это пластиковые диски с напылением тонкого слоя светоотражающего материала, на поверхности которых информация записана с помощью лазерного луча. Лазерные диски являются наиболее популярными съемными носителями информации. При размерах 12 см в диаметре их ёмкость достигает 700 Мб. В настоящее время все более популярным становится

формат компакт–дисков DVD-ROM, позволяющий при тех же размерах носителя разместить информацию объемом 4,3 Гб. Кроме того, доступными массовому покупателю стали устройства записи на компакт диски. Данная технология получила название CD–RW и DVD–RW соответственно.

Различные виды памяти имеют свои достоинства и недостатки. Так, внутренняя память имеет хорошее быстродействие, но ограниченный объем. Внешняя память, наоборот, имеет низкое быстродействие, но неограниченный объем. Производителям и пользователям компьютеров приходится искать компромисс между объемом памяти, скоростью доступа и ценой компьютера, так комбинируя разные виды памяти, чтобы компьютер работал оптимально. В любом случае, объем оперативной памяти является основной характеристикой ЭВМ и определяет производительность компьютера.

Сектор. Информация на магнитных дисках записывается в виде очень узких концентрических колец, расположенных очень близко друг к другу. Эти узкие кольца разбиваются на маленькие *сектора* — минимальные единицы физической, аппаратной записи. Длина сектора строго фиксирована и обычно составляет 512 байт.

На рис. 2.1 схематически изображена разбивка диска на секторы.

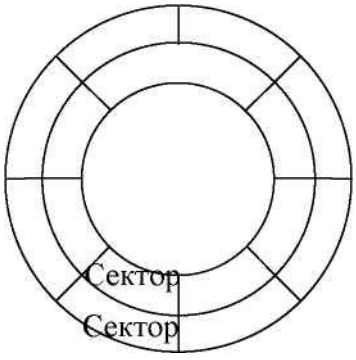


Рис. 2.1. Схематическое изображение секторов на диске

Кластер. В свою очередь сектора объединяются в последовательные цепочки — *кластеры* — минимальные единицы, кванты логической,

программной записи на магнитные диски. Длина кластера тоже фиксирована на одном диске, но эта величина зависит от его объема.

Прямой и последовательный доступ. Вся описанная память являются памятью прямого, или произвольного, доступа. При работе с такой памятью компьютер обращается непосредственно сразу к нужной единице информации в памяти (на магнитных дисках это кластеры или секторы).

Память последовательного доступа представляют накопители на магнитной ленте. Устройство, работающее с магнитной лентой, вынуждено прокручивать, перематывать все предыдущие участки магнитной ленты, чтобы добраться до нужных данных.

2.2. Основные компоненты персонального компьютера

2.2.1. Системный блок

Главная часть современного персонального компьютера (ПК) — ***системный блок***, который содержит:

1) *системную, или материнскую, печатную плату;*
2) *процессор,* находящийся на этой плате и выполняющий основные вычисления компьютера, в том числе выполнение компьютерных программ;
3) *оперативную память,* также находящуюся на системной плате, тесно связанную с процессором и хранящей код выполняемых программ.

На системной плате имеются также гнезда, или слоты, для подключения к компьютеру других устройств, находящихся вне системного блока.

Конструктивно в системном блоке также находятся жесткий диск, дисковод для дискет и дисковод для компактов. Все устройства систем-

ного блока подключены к *блоку питания,* соединенному с электрической сетью.

2.2.2. Периферия

Компьютерная периферия, или просто периферия,— это все компьютерные устройства, не входящие в состав системного блока.

Устройство ввода — устройство, позволяющее вводить данные в компьютер или управлять им. Устройствами ввода являются клавиатура и мышь, а также сканер, микрофон и др.

Устройство вывода выводит информацию из компьютера. Это монитор, принтер, звуковые колонки и др.

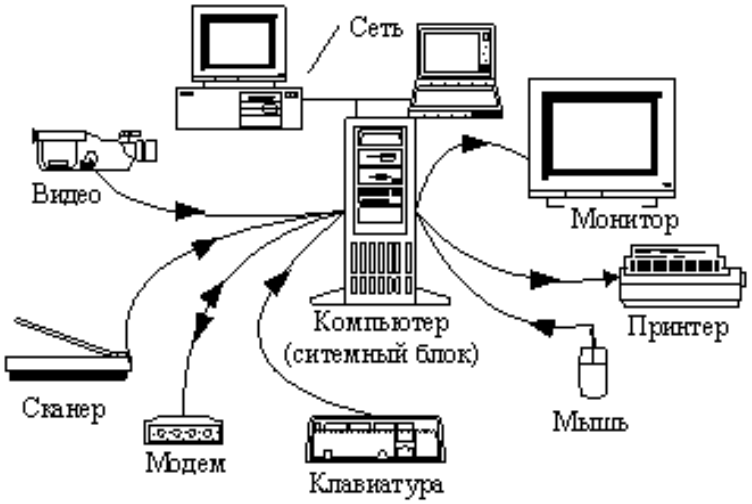


Рис. .2.2. Устройства ввода–вывода

2.2.2.1. Клавиатура

Клавиатура имеет более 100 клавиш, служащими для ввода текстов и управления компьютером. Клавиатура может иметь несколько языковых раскладок, т. е. может использовать свои клавиши для ввода букв разных алфавитов: русского, английского и т. п.

Рассмотрим ASCII (см. Приложение 1) коды стандартного набора символов. Они состоят из символов следующих групп.

– Прописных (больших) и строчных (маленьких) букв современного латинского алфавита, содержащего 26 символов.

– 10 цифр.

– 33 знаков препинания и специальных знаков.

Всего получаем $26 \cdot 2 + 10 + 33 = 95$ символов. В кодовых таблицах они кодируются числами от 32 до 126 включительно: 126 – 31 = 95.

На клавиатуре имеется 47 *алфавитно-цифровых клавиш* — буквы, цифры, знаки препинания и *специальные символы.* На каждой в режиме английского языка набирается по два символа, и вместе с клавишей пробела получаем эти же 95 символов.

2.2.2.2. Мышь

Мышь, или манипулятор «мышь» — это устройство для управления компьютером и ввода данных. На экране монитора мыши соответствует *указатель мыши,* движение которого по экрану управляется движением мыши по коврику. *У*правление различными объектами на экране монитора осуществляется наведением указателя мыши на соответствующий объект и нажатием при этом управляющих кнопок манипулятора «мышь».

2.2.2.3. Монитор

Монитор, или дисплей,— устройство вывода компьютером визуальных данных. На утомляемость глаз влияет *частота обновления экрана,* т е. количество кадров в секунду. Минимальная приемлемая частота 85 Гц.

Изображение на экране монитора состоит из цветных пикселей. ***Пиксель*** — это единица цвета монитора, точка–зерно, состоящая из то-

чек трех цветов, в сумме дающих цвет пикселя. Качество изображения зависит от разрешения аппаратуры.

Разрешение аппаратуры — количество точек на единицу длины, измеряемое по вертикали или горизонтали. В качестве длины всегда выступает дюйм, равный примерно 2,5 см. Обычно разрешение аппаратуры по обоим направлениям, горизонтали и вертикали, одинаковое, поэтому для обозначения разрешения используется только одно число.

Точка по-английски «*dot*», дюйм — «*inch*», точек на дюйм — «*dot per inch*», поэтому выражение «точек на дюйм» сокращается как *«dpi»*.

Итак, *разрешение* — это характеристика картинки на плоскости.

2.2.2.4. Сканер, принтер

Сканер — устройство оптического ввода информации, ее *сканирования,* фотографирования, служащее для копирования картинок окружающей действительности в компьютер. При работе сканера в компьютере создается графический объект — копия реальной картинки.

Планшетный сканер размещается на столе. При сканировании считывающее устройство перемещается вдоль планшета. При этом на стекло планшета кладется лист носителя или книга. При наличии дополнительной приставки возможно автоподача листов с изображениями.

Рулонный, или барабанный, сканер протягивает лист бумаги вдоль оптического считывателя, подобно принтеру. Только принтер печатает на бумаге, а рулонный сканер сканирует, фотографирует лист бумаги. В него можно вводить информацию с рулонного носителя или осуществлять автоподачу листов одного за другим.

Проекционный сканер снимает окружающие предметы, как фотоаппарат или телекамера. В отличие от предыдущих видов сканера в проекционном сканере нет движущихся частей. *Цифровые камера* и *видеокамера* представляют собой разновидности проекционного сканера.

Результатом цифрового сканирования является матрица из пикселей, расположенных равномерно по вертикали и горизонтали. Эта картинка имеет размеры и разрешение подобно изображению на экране монитора.

Специальной компьютерной программой распознавания текстов возможно преобразование текстовой информации, находящейся на картинке, снятой сканером из графического в текстовый формат.

Принтер — устройство вывода информации, используемое для печати данных на твердом носителе — бумаге или пленке.

Матричный принтер появился первым и назван в противоположность векторному устройству — *графопостроителю, или плоттеру.* Матричный принтер переносит изображение на бумагу, печатая точки, равномерно расположенные по вертикали и горизонтали. Графопостроитель рисует изображение специальными перьями, фломастерами.

Матричный принтер печатает стальными иголками, бьющими по пишущей ленте, пропитанной типографской краской. Поэтому его более правильное название — *игольчатый принтер.*

Затем появились *лазерные* принтеры, печатающие точками на твердом носителе (матричный принцип) при помощи мелкого черного порошка, который после нанесения на бумагу или пленку вплавляется в нее.

Струйные принтеры печатают краской разных цветов, которая точечными капельками (опять матричный принцип) напыляется на бумагу или другой твердый носитель, разбрызгиваясь через специальные микросопла.

2.2.3. Задания.

1. Графическая картинка занимает на экране площадь 15x15 $см^2$ и содержит 360 тысяч пикселей. Найти разрешение экрана.

2. Графическая картинка имеет ширину 600 пикселей. На экране она имеет ширину 15 см, а при печати — 2,5 см. Определить разрешения экрана и принтера.

3. Объем дискеты составляет 1,44 Мб. Сколько файлов можно записать на дискету, если длина каждого файла равна 250 Кб?

4. Страница, набранная в современном текстовом редакторе, занимает 4 Кб памяти. Сколько 100 страничных документов может поместиться на дискете, объем которой 1,44 Мб?

5. Объем дискеты составляет 1,44 Мб. Величина сектора дискеты равна величине ее кластера и равна 512 байт. Сколько кластеров на дискете?

6. Объем сектора составляет 512 байт. Максимальное количество секторов в кластере 64. Максимальное количество кластеров на винчестере 2^{16}. Определить максимальную емкость такого винчестера.

2.3. Программы

Компьютер = аппаратура + программы. Аппаратура представляет собой «жесткую» часть компьютера и по–английски называется соответственно — *hard.* Программы — это «мягкая» часть компьютера, по-английски называется *soft.* Программы находятся в форме файлов. Итак, получаем эквивалентную формулу *компьютер = hard + soft.*

2.3.1. Интерфейс программы, его виды

Интерфейс компьютерной программы — это ее внешний вид на экране дисплея, включающий ее оформление, вид и расположение элементов управления работой этой программы.

Текстовый интерфейс состоит только из символов, каждый из которых находится в какой-то текстовой строке и столбце на экране мони-

тора. В этом интерфейсе экран разбит 25 строками и 80 столбцами на 2000 ячеек, в каждой из которых может находиться один символ.

Графический интерфейс гораздо богаче текстового, он состоит из окон и управляющих элементов, изображенных на экране. Здесь нет никаких текстовых ячеек, графическое изображение строится с точностью до пикселя.

2.3.2. Операционная система, ее компоненты

Программы делятся на операционные системы и прикладные программы.

Операционной системой (ОС) называется комплект программ, которые совместно управляют ресурсами системы и процессами, использующими эти ресурсы. Выполнение любой программы на компьютере происходит под управлением ОС.

Программы, из которых состоит ОС, делятся на следующие три категории.

1. *Ядро ОС,* выполняющее основные функции ОС (в основном загрузку ее компонентов и поддержку выполнения компьютерных программ, в том числе и этих компонентов).
2. *Программу управления файлами и директориями,* служащую для классификации и просмотра информации, с которой имеет дело пользователь на компьютере.
3. *Драйверы,* которые позволяют ОС работать с аппаратурой: периферийными устройствами (монитор, клавиатура, мышь, принтеры и т. д.) и устройствами, входящими в состав системного блока (видеокарта, жесткий диск и т. д.). Без драйверов невозможно функционирование никаких компьютерных устройств.

2.3.3. Виды операционных систем

В настоящее время, с появлением мощных компьютеров, широкое распространение получили два типа ОС. К первому типу относятся достаточно похожие ОС семейства *Windows* компании *Microsoft*. Они многозадачные и имеют многооконный графический интерфейс. На рынке персональных компьютеров с *Windows* конкурируют ОС типа *UNIX*. Это многозадачная многопользовательская ОС с командным интерфейсом. В настоящее время разработаны расширения *UNIX*, обеспечивающие многооконный графический интерфейс. *UNIX* развивалась в течение многих лет разными компаниями, но до недавнего времени она не использовалась на персональных компьютерах, т.к. требует очень мощного процессора, весьма дорога и сложна, её установка и эксплуатация требуют высокой квалификации. В последние годы ситуация изменилась. Компьютеры стали достаточно мощными, появилась некоммерческая, бесплатная версия системы *UNIX* для персональных компьютеров — система *Linux*. По мере роста популярности этой системы в ней появились дополнительные компоненты, облегчающие её установку и эксплуатацию. Немалую роль в росте популярности *Linux* сыграла мировая компьютерная сеть *Internet*. Хотя освоение *Linux* гораздо сложнее освоения систем типа *Windows, Linux* - более гибкая и в то же время бесплатная система, что и привлекает к ней многих пользователей.

Существуют и другие ОС. Известная компания *Apple* производит компьютеры *Macintosh* с современной ОС *MacOS*. Эти компьютеры используются преимущественно издателями и художниками. Фирма *IBM* производит ОС *OS/2*. Операционная система *OS/2* такого же класса надёжности и защиты, как и *Windows NT*.

2.3.4. Утилиты

Под управлением ОС на компьютерах работают прикладные программы, которыми пользуются пользователи.

Существуют также программы, занимающие промежуточное положение между прикладными программами и ОС — это ***утилиты,*** или вспомогательные программы. Большинство утилит поставляется вместе с ОС, также эти утилиты производят другие отдельные фирмы.

Рассмотрим два класса утилит, не входящих в состав ОС. Различные их реализации поставляются разными фирмами.

Архиватор — программа, которая используется для сокращения объема хранимой или передаваемой информации. Архиватор по алгоритмам сжатия кодирует исходные данные, уплотняя их. Эта уплотненная информация хранится или передается по назначению, и затем при необходимости может быть полностью восстановлена в прежнем объеме.

Результатом работы архиватора является *архив* — файл со сжатой информацией. Можно запаковывать не только файлы одного каталога, но и целое дерево, иерархию каталогов со всеми файлами.

Антивирусная программа, или антивирус,— программа для борьбы с компьютерными вирусами. *Компьютерный вирус, или вирус,*— компьютерная программа, которая не имеет своего выполняемого файла, а внедряется, самодописывается в файлы других программ. Все вирусы опасны для нормальной работы компьютера, даже и так называемые «безвредные», поскольку они все равно портят с непредсказуемыми последствиями код программ.

Чтобы вирус активизировался и заработал, он должен попасть в оперативную память компьютера как *программа.* Поэтому при копировании и передаче файла с вирусом, когда он попадает в память как пассивные *данные,* заражения новых файлов и памяти компьютера не происходит.

2.3.5. Прикладные программы

Прикладная программа, или приложение, позволяет пользователю делать то, ради чего он использует компьютер, т. е. применять компьютер в разных областях человеческой деятельности. Прикладная программа выполняется на компьютере под управлением ОС.

Прикладные программы, в свою очередь, можно разделить на два класса:

1) программы–автоматы;

2) программы–инструменты.

Программы-автоматы — это прикладные программы, где пользователь эксплуатирует алгоритмы и данные, а также способы классификации данных и их просмотра, созданные другими людьми.

С помощью этих программ пользователь не создает новой информации. Это самые легкие в использовании программы, не требующие никаких специальных знаний.

Приведем три разновидности программ-автоматов: обучающие, игры и базы знаний.

– ***Обучающие программы*** помогают пользователю обучиться какой-нибудь области знания (языки, набор на клавиатуре, математика и т. д.).

Современные обучающие программы обычно являются мультимедийными, включая не только звук и работу с микрофоном, но и отрывки из видеофильмов.

– ***Игры*** используются для отдыха за компьютером, спортивных соревнований, тренировки логического мышления, тренажерной тренировки определенных навыков и умений, а также обучения.

Различают следующие классы игр: логические, стратегические, квесты (бродилки), симуляторы, аркады (стрелялки).

– ***Базы знаний*** — самая разнообразная информация, организованная в логические структуры. Частный случай таких программ — *экспертные системы,* которые помогают специалистам обрабатывать специальные данные и делать заключения. Эти программы легче перечислить по областям знаний: медицинские, математические, статистические и т. д.

– ***Сайт*** — организация информации в пространстве Интернета, представляющая собой ряд связанных между собой страниц одной тематики.

Программы-инструменты — это прикладные программы, с помощью которых пользователь создает новую авторскую информацию, хранящуюся в соответствующих файлах.

Программы-инструменты также делятся на два класса:

1) ***редакторы*** — программы для создания, редактирования, просмотра и изменения новой информации, за исключением компьютерных программ;
2) ***системы программирования, или языки программирования*** — программы для создания компьютерных программ.

2.3.5.1. Текстовые редакторы

Текстовые редакторы служат для создания разнообразных текстов на естественных и компьютерных языках.

Развитые текстовые редакторы с возможностями форматирования текста называются *текстовыми процессорами.*

Мощные текстовые процессоры используются только для верстки книг и называются *издательскими системами.*

2.3.5.2. Графические и мультимедийные редакторы

Графические редакторы обрабатывают графическую информацию, состоящую из пикселей или формул, позволяют добавлять в нее графические эффекты. Они также обрабатывают анимационную информацию, состоящую из последовательных кадров графической информации..

Мультимедийные редакторы имеют дело с полной коллекцией мультимедиа, в том числе звуком и видео. *Звуковые редакторы* позволяют визуально просматривать оцифрованный звук, редактировать и прослушивать его. *Видео-редакторы* занимаются с оцифрованным видео: осуществляют покадровый просмотр, редактирование и добавление видео–эффектов, монтаж и озвучивание видео-информации.

2.3.5.3. Редакторы баз данных

Редакторы баз данных, или системы управления базами данных (СУБД), занимаются базами данных (БД), т. е. самой разнообразной информацией, организованной в логические структуры.

Их разновидностью являются ***табличные редакторы,*** которые создают и обрабатывают числовые таблицы, в которых хранится исключительно числовая информация и формулы для обработки этих чисел.

Еще одной разновидностью СУБД являются специальные программы, которые легче перечислять по областям знаний: математические, статистические, бухгалтерские и т. д. Эти программы накапливают и редактируют данные в тех специальных областях знания, где они применяются.

2.3.5.4. Языки программирования

Системы программирования, или языки программирования — это прикладные программы, которые позволяют программисту создавать любые компьютерные программы.

Этими компьютерными программами являются:

1. прикладные программы, в том числе языки программирования;
2. утилиты;
3. вирусы;
4. операционные системы.

Самые распространенные языки программирования: Бейсик, Паскаль, Си.

2.4. Файл и дерево каталогов

Файл — это форма организации информация на компьютере. Обычно файл — это именованная область компьютерного диска, т. е. файл имеет имя и содержание. Почти все программы и все данные хранятся в файлах.

Содержанием файла является закодированный результат работы прикладной программы. По виду информации, которую хранят файлы, они делятся на выполняемые, т. е. программы, и не выполняемые, т.е. текстовые, графические, звуковые, и т. д. Объем файлов измеряется в байтах, килобайтах и мегабайтах.

Каждый файл имеет ***имя,*** составляемое по особым правилам. Стандартное имя, которое «понимает» ОС на любом компьютере и в любой стране, отвечает следующим требованиям.

1. Имя файла имеет длину от одного до 8 символов. Имя файла составляется так, чтобы передавать его содержание; например: lekcija.

2. Символы имени файла являются либо латинскими буквами, либо цифрами, либо некоторыми специальными символами, в число которых пробел ни в коем случае не входит.

3. Имя файла может иметь *расширение* длиной от 1 до трех символов из того же комплекта, причем, если расширение есть, оно отделяется от основного имени файла точкой. Расширение файла составляется так,

чтобы отражать тип файла; например, lekcija.doc — текстовый файл.

Таким образом, максимальная длина имени файла, отвечающего указанному стандарту, равна 12 символов: 8 символов основное имя, 1 точка, 3 символа расширение имени файла.

В современных русифицированных компьютерных системах имя файла может иметь большую длину и включать русские буквы.

2.4.1. Имена дисков

Диски на персональных компьютерах именуются отдельными латинскими буквами и дополняются двоеточием, чтобы их можно было отличить от имен файлов. Например: *A*:, *C*: или *D*:.

Обычно на ПК диски имеют следующие имена:

1) дискеты имеют имя *A*:;
2) жесткие диски имеют имя *C*:;
3) дисководы компакт-дисков имеют имя *D*:;
4) имена сетевых дисков, находящихся на другом компьютере, начинаются с имени *F*: и заканчиваются именем *Z*:.

2.4.2. Каталог

На жестких дисках файлов очень много — десятки тысяч. Чтобы работать с таким большим количеством объектов, их необходимо классифицировать, распределить по группам, создать из них структуру. Структура файлов на дисках всегда имеет вид дерева.

Файлы объединены по своему назначению в логические группы (см. рис. 11), расположенные на дисках компьютера и имеющие свои имена. Эти группы называются *каталогами, или директориями, или папками,* а их имена — именами каталогов.

2.4.3. Дерево каталогов, надкаталог

Каталог нижнего уровня, в свою очередь, объединяются в группы — каталоги верхнего уровня, или *надкаталоги (наддиректории, надпапки)* (рис. 2.1).

Эти директории второго уровня снизу также объединяются в директории и т. д. до самой верхней *корневой директории (каталога, папки)*.

Корневая директория именуются по названию диска с добавлениям символа обратная наклонная черта \, например, *C*:\ или *D*:\ (рис. 2.1).

Вся эта иерархия называется *деревом директорий (каталогов, папок)*.

Часть дерева, у которой какая-то директория является самой верхней, называется *поддеревом*. Эта самая верхняя директория является корневой директорией этого поддерева.

На рис. 2.1 изображен пример дерева директорий. Прямоугольниками обозначены директории, кружками — файлы, в треугольники заключены примеры поддеревьев. Графически дерево директорий изображается в виде дерева, растущего корнем вверх и ветвями вниз, файлы — листья на дереве.

2.4.4. Создание дерева каталогов, подкаталог

При создании дерева каталогов процесс создания каталогов идет сверху вниз: сначала на диске в корневой каталоге создаются каталоги, называемые *подкаталогами (поддиректориями, подпапками),*— это каталоги первого уровня.

Затем при необходимости в каталогах первого уровня создаются подкаталоги — каталоги второго уровня, и т. д. до тех пор, пока вся информация не будет структурирована, классифицирована в виде дерева.

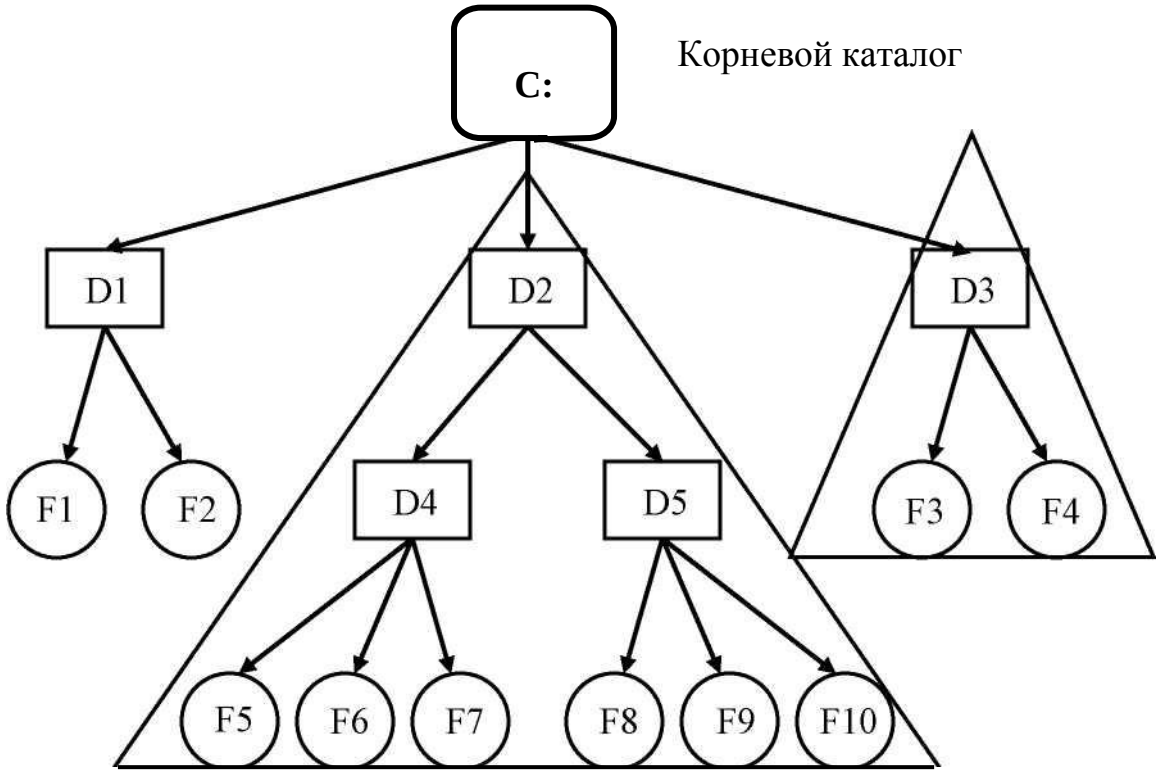


Рис. 2.1. Дерево каталогов.

2.4.5. Добавление новой информации

При появлении новой информации она добавляется:

1) либо в виде отдельного файла или файлов в какую-то уже существующую директорию;

2) либо в какой-нибудь, соответствующей по смыслу, директории создается еще одна новая поддиректория, в которую и записывается поступившая или созданная информация в виде поддерева, для которого эта новая поддиректория является корневой.

2.4.6. Имена и содержание каталогов

Имена каталогов составляются по тем же самым правилам, что и имена файлов. Обычно имена каталогов не имеют расширения. Содержанием каталога является:

1) список имен объектов (файлов и каталогов), которые входят в него;

2) указатель на вышестоящий каталог, надкаталог, которому принадлежит текущий каталог.

2.4.7. Задания

1. Пусть имена файлов можно составлять только из двух цифр 0 и 1. Сколько можно составить имен файлов без расширения, содержащих не более чем два символа?

2. Тот же вопрос, что и в упр. 1, но файлы могут иметь расширение.

3. На диске *C*: находятся два каталога *A* и *B* и один файл *1.doc*. Нарисуйте в виде дерева все 9 вариантов их взаимного расположения. Сколько различных поддеревьев можно выделить в каждом случае?

4. На диске *C*: находятся два каталога *A* и *B* и два файла *1.doc* и *2.doc*. Нарисуйте в виде дерева все 27 вариантов их взаимного расположения. Сколько различных поддеревьев можно выделить в каждом случае?

3. ИНФОРМАЦИЯ. КОДИРОВАНИЕ ИНФОРМАЦИИ

3.1. Понятие «информация» и свойства информации

Понятие «информация». Слово «информация» происходит от латинского слова informatio, что в переводе означает сведение, разъяснение, ознакомление. Понятие «информация» является базовым в курсе информатики, невозможно дать его определение через другие, более «простые» понятия. В геометрии, например, невозможно выразить содержание базовых понятий «точка», «луч», «плоскость» через более простые понятия. Содержание основных, базовых понятий в любой науке должно быть пояснено на примерах или выявлено путем их сопоставления с содержанием других понятий.

В случае с понятием «информация» проблема его определения еще более сложная, так как оно является общенаучным понятием. Данное понятие используется в различных науках (информатике, кибернетике, биологии, физике и др.), при этом в каждой науке понятие «информация» связано с различными системами понятий.

Информация в физике. В физике мерой беспорядка, хаоса для термодинамической системы является энтропия системы, тогда как информация (антиэнтропия) является мерой упорядоченности и сложности системы. По мере увеличения сложности системы величина энтропии уменьшается, и величина информации увеличивается. Процесс увеличения информации характерен для открытых, обменивающихся веществом и энергией с окружающей средой, саморазвивающихся систем живой природы (белковых молекул, организмов, популяций животных и так далее).

Таким образом, в физике информация рассматривается как антиэнтропия или энтропия с обратным знаком.

Информация в биологии. В биологии, которая изучает живую природу, понятие «информация» связывается с целесообразным поведением

живых организмов. Такое поведение строится на основе получения и использования организмом информации об окружающей среде.

Понятие «информация» в биологии используется также в связи с исследованиями механизмов наследственности. Генетическая информация передается по наследству и хранится во всех клетках живых организмов. Гены представляют собой сложные молекулярные структуры, содержащие информацию о строении живых организмов. Последнее обстоятельство позволило проводить научные эксперименты по клонированию, то есть созданию точных копий организмов из одной клетки.

Информация в кибернетике. В кибернетике (науке об управлении) понятие «информация» связано с процессами управления в сложных системах (живых организмах или технических устройствах). Жизнедеятельность любого организма или нормальное функционирование технического устройства зависит от процессов управления, благодаря которым поддерживаются в необходимых пределах значения их параметров. Процессы управления включают в себя получение, хранение, преобразование и передачу информации.

Социально значимые свойства информации. Человек — существо социальное, для общения с другими людьми он должен обмениваться с ними информацией, причем обмен информацией всегда производится на определенном языке — русском, английском и так далее. Участники дискуссии должны владеть тем языком, на котором ведется общение, тогда информация будет понятной всем участникам Обмена информацией.

Информация должна быть полезной, тогда дискуссия приобретает практическую ценность. Бесполезная информация создает информационный шум, который затрудняет восприятие полезной информации. Примерами передачи и получения бесполезной информации могут служить некоторые конференции и чаты в Интернете.

Широко известен термин «средства массовой информации» (газеты, радио, телевидение), которые доводят информацию до каждого члена

общества. Такая информация должна быть *достоверной* и *актуальной.* Недостоверная информация вводит членов общества в заблуждение и может быть причиной возникновения социальных потрясений. Неактуальная информация бесполезна и поэтому никто, кроме историков, не читает прошлогодних газет.

Для того чтобы человек мог правильно ориентироваться в окружающем мире, информация должна быть *полной* и *точной.* Задача получения полной и точной информации стоит перед наукой. Овладение научными знаниями в процессе обучения позволяют человеку получить полную и точную информацию о природе, обществе и технике.

3.2. Количество информации

Информация и знания. Человек получает информацию из окружающего мира с помощью органов чувств, анализирует ее и выявляет существенные закономерности с помощью мышления, хранит полученную информацию в памяти. Процесс систематического научного познания окружающего мира приводит к накоплению информации в форме знаний (фактов, научных теорий и так далее). Таким образом, с точки зрения процесса познания информация может рассматриваться как *знания.*

Процесс познания можно наглядно изобразить в виде расширяющегося круга знания (такой способ придумали еще древние греки). Вне этого круга лежит область незнания, а окружность является границей между знанием и незнанием. Парадокс состоит в том, что чем большим объемом знаний обладает человек (чем шире круг знаний), тем больше он ощущает недостаток знаний (тем больше граница нашего незнания, мерой которого в этой модели является длина окружности).

Так, объем знаний выпускника школы гораздо больше, чем объем знаний первоклассника, однако и граница его незнания существенно

больше. Действительно, первоклассник ничего не знает о законах физики и поэтому не осознает недостаточности своих знаний, тогда как выпускник школы. При подготовке к экзаменам по физике может обнаружить, что существуют физические законы, которые он не знает или не понимает.

Информацию, которую получает человек, можно считать мерой уменьшения неопределенности знаний. Если некоторое сообщение приводит к уменьшению неопределенности наших знаний, то можно говорить, что такое сообщение содержит информацию.

Например, после сдачи экзамена по информатике вы мучаетесь неопределенностью, вы не знаете, какую оценку получили. Наконец, экзаменационная комиссия объявляет результаты экзамена, и вы получаете сообщение, которое приносит полную определенность, теперь вы знаете свою оценку. Происходит переход от незнания к полному знанию, значит, сообщение экзаменационной комиссии содержит информацию.

Уменьшение неопределенности знаний. Подход к информации как мере уменьшения неопределенности знаний позволяет количественно измерять информацию, что чрезвычайно важно для информатики. Рассмотрим вопрос об определении количества информации более подробно на конкретных примерах.

Пусть у нас имеется монета, которую мы бросаем на ровную поверхность. С равной вероятностью произойдет одно из двух возможных событий — монета окажется в одном из двух положений: «орел» или «решка».

Можно говорить, что события равновероятны, если при возрастающем числе опытов количества выпадений «орла» и «решки» постепенно сближаются. Например, если мы бросим монету 10 раз, то «орел» может выпасть 7 раз, а решка — 3 раза, если бросим монету 100 раз, то «орел» может выпасть 60 раз, а «решка» — 40 раз, если бросим монету 1000 раз, то «орел» может выпасть 520 раз, а «решка» — 480 и так далее.

В итоге при очень большой серии опытов количества выпадений «орла» и «решки» практически сравняются.

Перед броском существует неопределенность наших знаний (возможны два события), и, как упадет монета, предсказать невозможно. После броска наступает полная определенность, так как мы видим (получаем зрительное сообщение), что монета в данный момент находится в определенном положении (например, «орел»). Это сообщение приводит к уменьшению неопределенности наших знаний в два раза, так как до броска мы имели два вероятных события, а после броска — только одно, то есть в два раза меньше (рис. 3.1).

Возможные события	*Происшедшие события*
	
	

Рис. 3.1. Возможные и произошедшее события

В окружающей действительности достаточно часто встречаются ситуации, когда может произойти некоторое количество равновероятных событий. Так, при бросании равносторонней четырехгранной пирамиды существуют 4 равновероятных события, а при бросании шестигранного игрального кубика — 6 равновероятных событий.

Чем больше количество возможных событий, тем больше начальная неопределенность и соответственно тем большее количество информации будет содержать сообщение о результатах опыта.

Единицы измерения количества информации. Для количественного выражения любой величины необходимо определить единицу измерения. Так, для измерения длины в качестве единицы выбран метр, для измерения массы — килограмм и так далее. Аналогично, для определения количества информации необходимо ввести единицу измерения.

За единицу количества информации принимается такое количество информации, которое содержит сообщение, уменьшающее неопределенность в два раза. Такая единица названа ***«бит».***

Если вернуться к опыту с бросанием монеты, то здесь неопределенность как раз уменьшается в два раза и, следовательно, полученное количество информации равно 1 биту.

Количество возможных событий и количество информации. Существует формула, которая связывает между собой количество возможных событий N и количество информации: $N = 2^I$

По этой формуле можно легко определить количество возможных событий, если известно количество информации. Например, если мы получили 4 бита информации, то количество возможных событий составляло:

$N = 2^4 = 16$.

Наоборот, для определения количества информации, если известно количество событий, необходимо решить показательное уравнение относительно I. Например, в игре «Крестики–нолики» на поле 8×8 перед первым ходом существует 64 возможных события (64 различных варианта расположения «крестика»), тогда уравнение принимает вид:

$64 = 2^I$

Так как $64 = 2^6$, то получим:

$2^6 = 2^I$.

Таким образом, I = 6 битов, то есть количество информации, полученное вторым игроком после первого хода первого игрока, составляет 6 битов.

3.2.1. Вопросы и задания

1.Приведите примеры уменьшения неопределенности знаний после получения информации о произошедшем событии.

2.В чем состоит неопределенность знаний в опыте по бросанию монеты?

3.Как зависит количество информации от количества возможных событий?

4.Какое количество информации получит второй игрок после первого хода первого игрока в игре в «Крестики-нолики» на поле размером 4×4?

5.Каково было количество возможных событий, если после реализации одного из них мы получили количество информации, равное 3 битам? 7 битам?

3.3. Алфавитный подход к определению количества информации

При определении количества информации на основе уменьшения неопределенности наших знаний мы рассматриваем информацию с точки зрения содержания, ее понятности и новизны для человека. С этой точки зрения в опыте по бросанию монеты одинаковое количество информации содержится и в зрительном образе упавшей монеты, и в коротком сообщении «Орел», и в длинной фразе «Монета упала на поверхность земли той стороной вверх, на которой изображен орел».

Однако при хранении и передаче информации с помощью технических устройств целесообразно отвлечься от содержания информации и рассматривать ее как последовательность знаков (букв, цифр, кодов цветов точек изображения и так далее).

Набор символов знаковой системы (алфавит) можно рассматривать как различные возможные состояния (события). Тогда, если считать, что

появление символов в сообщении равновероятно, по формуле $N = 2^I$ можно рассчитать, какое количество информации несет каждый символ.

Так, в русском алфавите, если не использовать букву ё, количество событий (букв) будет равно 32. Тогда: $32 = 2^I$, откуда $I = 5$ битов.

Каждый символ несет 5 битов информации (его информационная емкость равна 5 битов). Количество информации в сообщении можно подсчитать, умножив количество информации, которое несет один символ, на количествосимволов.

Вопрос:

1. Пусть две книги на русском и китайском языках содержат одинаковое количество знаков. В какой книге содержится большее количество информации с точки зрения алфавитного подхода?

3.4. Представление и кодирование информации

3.4.1. Язык как знаковая система

Для обмена информацией с другими людьми человек использует *естественные языки* (русский, английский, китайский и др.), то есть информация представляется с помощью естественных языков. В основе языка лежит алфавит, то есть набор символов (знаков), которые человек различает по их начертанию. В основе русского языка лежит кириллица, содержащая 33 знака, английский язык использует латиницу (26 знаков), китайский язык использует алфавит из десятков тысяч знаков (иероглифов).

Последовательности символов алфавита в соответствии с правилами *грамматики* образуют основные объекты языка — слова. Правила, согласно которым образуются предложения из слов данного языка, называются *синтаксисом*. Необходимо отметить, что в естественных языках грамматика и синтаксис языка формулируются с помощью большого количества правил, из которых существуют исключения, так как такие

правила складывались исторически.

Наряду с естественными языками были разработаны *формальные языки* (системы счисления, язык алгебры, языки программирования и др.). Основное отличие формальных языков от естественных состоит в наличии строгих правил грамматики и синтаксиса.

Например, системы счисления можно рассматривать как формальные языки, имеющие алфавит (цифры) и позволяющие не только именовать и записывать объекты (числа), но и выполнять над ними арифметические операции по строго определенным правилам.

Некоторые языки используют в качестве знаков не буквы и цифры, а другие символы, например химические формулы, ноты, изображения элементов электрических или логических схем, дорожные знаки, точки и тире (код азбуки Морзе) и др.

Представление информации может осуществляться с помощью языков, которые являются знаковыми системами. Каждая знаковая система строится на основе определенного алфавита и правил выполнения операций над знаками.

Знаки могут иметь различную физическую природу. Например, для представления информации с использованием языка в письменной форме используются знаки, которые являются изображениями на бумаге или других носителях, в устной речи в качестве знаков языка используются различные звуки (фонемы), а при обработке текста на компьютере знаки представляются в форме последовательностей электрических импульсов (компьютерных кодов).

Вопросы

1.Чем различаются естественные и формальные языки?

2.Какова может быть физическая природа знаков?

3.4.2. Кодирование информации

Представление информации происходит в различных формах в процессе восприятия окружающей среды живыми организмами и человеком, в процессах обмена информацией между человеком и человеком, человеком и компьютером, компьютером и компьютером и так далее. Преобразование информации из одной формы представления (знаковой системы) в другую называется *кодированием.*

Средством кодирования служит таблица соответствия знаковых систем, которая устанавливает взаимно однозначное соответствие между знаками или группами знаков двух различных знаковых систем.

В процессе обмена информацией часто приходится производить операции *кодирования* и *декодирования* информации. При вводе знака алфавита в компьютер путем нажатия соответствующей клавиши на клавиатуре происходит кодирование знака, то есть преобразование его в компьютерный код. При выводе знака на экран монитора или принтер происходит обратный процесс — декодирование, когда из компьютерного кода знак преобразуется в его графическое изображение.

Кодирование — это операция преобразования знаков или групп знаков одной знаковой системы в знаки или группы знаков другой знаковой системы.

3.4.3. Представление информации в компьютере

Для представления информации в компьютере используются 2 символа — ноль и единица (0 и 1), аналогично тому, как в азбуке Морзе используются точка и тире. Действительно, закодировав привычные человеку символы (буквы, цифры, знаки) в виде нулей и единиц (или точек и тире), можно составить, передать и сохранить любое сообщение.

Это связано с тем, что информацию, представленную в таком виде, легко технически смоделировать, например, в виде электрических сиг-

налов. Если в какой-то момент времени по проводнику идет ток, то по нему передается единица, если тока нет — ноль. Аналогично, если направление магнитного поля на каком-то участке поверхности магнитного диска одно — на этом участке записан ноль, другое — единица. Если определенный участок поверхности оптического диска отражает лазерный луч — на нем записан ноль, не отражает — единица. Оперативная память состоит из очень большого числа триггеров — электронных схем, состоящих из двух транзисторов. Триггер может сколь угодно долго находиться в одном из двух состояний — когда один транзистор открыт, а другой закрыт, или наоборот. Одно состояние обозначается нулем, а другое единицей.

Как отмечалось выше, объем информации, необходимый для запоминания одного из двух символов 0 или 1, называется 1 бит (англ. Binary digit — двоичная единица). Таким образом, 1 бит — минимально возможный объем информации.

Итак, если у нас есть один бит, то с его помощью мы можем закодировать один из двух символов — либо 0, либо 1.

Если же есть 2 бита, то из них можно составить один из четырех вариантов кодов: 00, 01, 10, 11.

Если есть 3 бита — один из восьми: 000, 001, 010, 100, 110, 101, 011, 111.

Закономерность очевидна:

1 бит – 2 варианта,

2 бита – 4 варианта,

3 бита – 8 вариантов;

Продолжая дальше, получим:

4 бита – 16 вариантов,

5 бит – 32 варианта,

6 бит – 64 варианта,

7 бит – 128 вариантов,

8 бит – 256 вариантов,

9 бит – 512 вариантов,

10 бит – 1024 варианта,

.....

N бит – 2 в степени N вариантов.

В обычной жизни нам достаточно 150–160 стандартных символов (больших и маленьких русских и латинских букв, цифр, знаков препинания, арифметических действий и т.п.). Если каждому из них будет соответствовать свой код из нулей и единиц, то 7 бит для этого будет недостаточно (7 бит позволят закодировать только 128 различных символов), поэтому используют 8 бит.

Кодирование одного привычного человеку символа в ЭВМ 8–ю битами позволяет закодировать 256 различных символов.

Стандартный набор из 256 символов называется ASCII («Американский Стандартный Код для Обмена Информацией» — англ. *American Standart Code for Information Interchange*) (см. Приложение 1).

Он включает в себя большие и маленькие русские и латинские буквы, цифры, знаки препинания и арифметических действий и т.п., причем каждому символу ASCII соответствует 8–битовый двоичный код, например:

A – 01000001,

B – 01000010,

C – 01000011,

D – 01000100,

и т.д.

Таким образом, если человек создает текстовый файл и записывает его на диск, то на самом деле каждый введенный человеком символ хранится в памяти компьютера в виде набора из восьми нулей и единиц. При выводе этого текста на экран или на бумагу специальные схемы — знакогенераторы видеоадаптера (устройства, управляющего работой

дисплея) или принтера образуют в соответствии с этими кодами изображения соответствующих символов.

Набор ASCII был разработан в США Американским Национальным Институтом Стандартов (ANSI), но может быть использован и в других странах, поскольку вторая половина из 256 стандартных символов, т.е. 128 символов, могут быть с помощью специальных программ заменены на другие символы, в частности на символы национального алфавита, в нашем случае — буквы кириллицы. Поэтому, например, передавать по электронной почте за границу тексты, содержащие русские буквы, бессмысленно. В англоязычных странах на экране дисплея вместо русской буквы Ь будет высвечиваться символ английского фунта стерлинга, вместо буквы р — греческая буква альфа, вместо буквы л — одна вторая и т.д.

Как уже отмечалось, минимальной единицей измерения количества информации является бит. Следующей же по величине единицей является байт, причем

1 байт = 2^3 бит = 8 бит

В информатике система образования кратных единиц измерения количества информации несколько отличается от принятых в большинстве наук. Традиционные метрические системы единиц, например Международная система единиц СИ, в качестве множителей кратных единиц используют коэффициент 10^n, где n = 3, 6, 9 и так далее, что соответствует десятичным приставкам Кило (10^3), Мега ($10^{6)}$, Гига ($10^{9)}$ и так далее.

Компьютер оперирует числами не в десятичной, а в двоичной системе счисления, поэтому в кратных единицах измерения количества информации используется коэффициент 2^n.

1 килобайт = 2^{10} байт = 1024 байта,

1 мегабайт = 2^{10} Кбайт = 1024 килобайтам,

1 гигабайт = 2^{10} Мбайт = 1024 мегабайтам,

1 терабайт = 2^{10} Гбайт = 1024 гигабайтам.

Обратите внимание, что в информатике смысл приставок кило– , мега– и других в общепринятом смысле выполняется не точно, а приближенно, поскольку соответствует увеличению не в 1000, а в 1024 раза.

Заполним таблицу с коэффициентами перевода производных единиц от байта. Например, 1 Мбайт = 2^{10} Килобайт, 1 Кбайт = 2^{-10} Мегабайт.

Таблица 3.1. Коэффициенты переводы производных единиц от байта

	Байт	Килобайт	Мегабайт	Гигабайт	Терабайт
байт	1	2^{-10}	2^{-20}	2^{-30}	2^{-40}
Кбайт	2^{10}	1	2^{-10}	2^{-20}	2^{-30}
Мбайт	2^{20}	2^{10}	1	2^{-10}	2^{-20}
Гбайт	2^{30}	2^{20}	2^{10}	1	2^{-10}
Тбайт	2^{40}	2^{30}	2^{20}	2^{10}	1

3.5. Системы счисления

Система счисления, или просто счисление, или нумерация,— набор конкретных знаков–цифр вместе с системой приемов записи, которая представляет числа этими цифрами.

Разные народы в разные времена использовали разные системы счисления. Следы древних систем счета встречаются и сегодня в культуре многих народов. К древнему Вавилону восходит деление часа на 60 минут и угла на 360 градусов. К Древнему Риму — традиция записывать в римской записи числа I, II, III и т. д. К англосаксам — счет дюжинами: в году 12 месяцев, в футе 12 дюймов, сутки делятся на 2 периода по 12 часов. Восточные арабы используют такую же систему счисления, что и в большинстве стран, но начертание цифр у них иное.

Таким образом, различные системы счисления отличаются друг от друга по следующим признакам:

1) разное начертание цифр, которые обозначают одни и те же числа;

2) разные способы записи чисел цифрами;

3) разное количество цифр.

По современным данным, развитые системы нумерации впервые появились в древнем Египте. Для записи чисел египтяне применяли иероглифы один, десять, сто, тысяча и т.д. Все остальные числа записывались с помощью этих иероглифов и операции сложения. Недостатки этой системы — невозможность записи больших чисел и громоздкость.

В конце концов, самой популярной системой счисления оказалась десятичная система. Десятичная система счисления пришла из Индии, где она появилась не позднее VI в. н. э. В ней всего 10 цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 но информацию несет не только цифра, но также и место позиция, на которой она стоит. В числе 444 три одинаковых цифры обозначают количество и единиц, и десятков, и сотен. А вот в числе 400 первая цифра обозначает число сотен, два 0 сами по себе вклад в число не дают, а нужны лишь для указания позиции цифры 4.

3.5.1. Основные понятия систем счисления

Система счисления — это совокупность правил и приемов записи чисел с помощью набора цифровых знаков. Количество цифр, необходимых для записи числа в системе, называют основанием системы счисления. Основание системы записывается в справа числа в нижнем индексе: 5_{10}; 1110110_2; $AF178_{16}$.

Различают два типа систем счисления:

– позиционные, когда значение каждой цифры числа определяется ее позицией в записи числа;

– непозиционные, когда значение цифры в числе не зависит от ее места в записи числа.

Примером непозиционной системы счисления является римская: числа IX, IV, XV и т.д. Примером позиционной системы счисления является десятичная система, используемая повседневно.

Любое целое число в позиционной системе можно записать в форме многочлена:

$$X_S = A_n A_{n-1} \ldots A_2 A_1 = A_n \cdot S^{n-1} + A_{n-1} \cdot S^{n-2} + \ldots + A_2 \cdot S^1 + A_1 \cdot S^0,$$

где S — основание системы счисления;

A_n — цифры числа, записанного в данной системе счисления;

n — количество разрядов числа.

Пример. Число 6293_{10} запишется в форме многочлена следующим образом:

$$6293_{10} = 6 \cdot 10^3 + 2 \cdot 10^2 + 9 \cdot 10^1 + 3 \cdot 10^0$$

Римская система счисления является непозиционной системой. В ней для записи чисел используются буквы латинского алфавита. При этом буква ***I*** всегда означает единицу, буква ***V*** — пять, ***X*** — десять, ***L*** — пятьдесят, ***C*** — сто, ***D*** — пятьсот, ***M*** — тысячу и т.д. Например, число 264 записывается в виде ***CCLXIV***.

Запись римскими цифрами. Натуральные числа, т. е. целые положительные числа (без нуля), можно записывать при помощи повторения римских цифр, используя три следующие правила.

1. ***Правило сложения***: если все цифры в числе по значению не возрастают, если считать слева направо, то они ***складываются.***

Например:

II = 2, ***VI*** = 6, ***XI*** = 11 — правильно;

IV = 6, ***XL*** = 60 — неправильно.

2. ***Правило вычитания***: 1) сначала во всех парах, где меньшая цифра стоит перед большей, ***вычитается*** меньшая цифра из большей; 2) затем полученные результаты вместе с оставшимися цифрами подпадают под принцип сложения и складываются.

Например:

IV = 4, ***XIV*** = 14, ***XXIX*** = 29 — правильно;

IVX = 6, ***IXX*** = 1 — неправильно.

Типичные примеры, иллюстрирующие общие правила записи чисел в римской система счисления, приведены в таблице 3.2.

Недостатком римской системы является отсутствие формальных правил записи чисел и, соответственно, арифметических действий с многозначными числами. По причине неудобства и большой сложности в настоящее время римская система счисления используется там, где это действительно удобно: в литературе (нумерация глав), в оформлении документов (серия паспорта, ценных бумаг и др.), в декоративных целях на циферблате часов и в ряде других случаев.

Десятичная система счисления – в настоящее время наиболее известная и используемая. Изобретение десятичной системы счисления относится к главным достижениям человеческой мысли. Без нее вряд ли могла существовать, а тем более возникнуть современная техника. Причина, по которой десятичная система счисления стала общепринятой, вовсе не математическая. Люди привыкли считать в десятичной системе счисления, потому что у них по 10 пальцев на руках.

Древнее изображение десятичных цифр (рис. 3.1) не случайно: каждая цифра обозначает число по количеству углов в ней. Например, 0 — углов нет, 1 — один угол, 2 — два угла и т.д. Написание десятичных цифр претерпело существенные изменения. Форма, которой мы пользуемся, установилась в XVI веке.

Таблица 3.2. Запись чисел в римской системе счисления

1	2	3	4
I	II	III	IV
5	6	7	8
V	VI	VII	VIII
9	10	11	13
IX	X	XI	XIII
18	19	22	34
XVIII	XIX	XXII	XXXIV
39	40	60	99
XXXIX	XL	LX	XCIX
200	438	649	999
CC	CDXXXVIII	DCXLIX	CMXCIX
1207	2045	3555	3900
MCCVII	MMXLV	MMMDLV	MMMCM
3678		3999	
MMMDCLXXVIII		MMMCMXCIX	

Десятичная система впервые появилась в Индии примерно в VI веке новой эры. Индийская нумерация использовала девять числовых символов и нуль для обозначения пустой позиции.

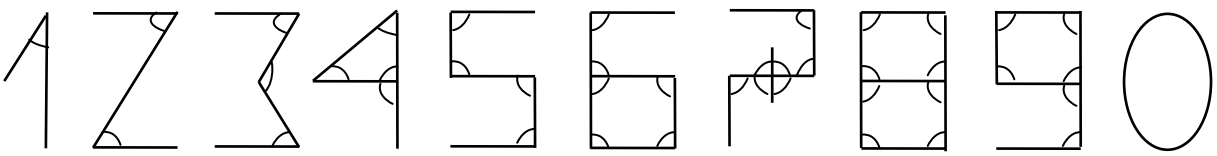


Рис.3.1. Изображение десятичных цифр

В ранних индийских рукописях, дошедших до нас, числа записывались в обратном порядке — наиболее значимая цифра ставилась справа. Но вскоре стало правилом располагать такую цифру с левой стороны.

Особое значение придавалось нулевому символу, который вводился для позиционной системы обозначений. Индийская нумерация, включая нуль, дошла и до нашего времени. В Европе индусские приёмы десятичной арифметики получили распространение в начале XIII в. благодаря работам итальянского математика Леонардо Пизанского (Фибоначчи). Европейцы заимствовали индийскую систему счисления у арабов, назвав ее арабской. Это исторически неправильное название удерживается и поныне.

Десятичная система использует десять цифр —– 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9, а также символы “+” и “–” для обозначения знака числа и запятую или точку для разделения целой и дробной частей числа.

В вычислительных машинах используется двоичная система счисления, её основание — число 2. Для записи чисел в этой системе используют только две цифры — 0 и 1. Вопреки распространенному заблуждению, двоичная система счисления была придумана не инженерами–конструкторами ЭВМ, а математиками и философами задолго до появления компьютеров, еще в XVII – XIX веках. Первое опубликованное обсуждение двоичной системы счисления принадлежит испанскому священнику Хуану Карамюэлю Лобковицу (1670 г.). Всеобщее внимание к этой системе привлекла статья немецкого математика Готфрида Вильгельма Лейбница, опубликованная в 1703 г. В ней пояснялись двоичные операции сложения, вычитания, умножения и деления. Лейбниц не рекомендовал использовать эту систему для практических вычислений, но подчёркивал её важность для теоретических исследований. Со временем двоичная система счисления становится хорошо известной и получает развитие.

Выбор двоичной системы для применения в вычислительной технике объясняется тем, что электронные элементы — триггеры, из которых состоят микросхемы ЭВМ, могут находиться только в двух рабочих состояниях.

С помощью двоичной системы кодирования можно зафиксировать любые данные и знания. Это легко понять, если вспомнить принцип кодирования и передачи информации с помощью азбуки Морзе. Телеграфист, используя только два символа этой азбуки — точки и тире, может передать практически любой текст.

Двоичная система удобна для компьютера, но неудобна для человека: числа получаются длинными и их трудно записывать и запоминать. Конечно, можно перевести число в десятичную систему и записывать в таком виде, а потом, когда понадобится перевести обратно, но все эти переводы трудоёмки. Поэтому применяются системы счисления, родственные двоичной — восьмеричная и шестнадцатеричная. Для записи чисел в этих системах требуется соответственно 8 и 16 цифр. В 16–ной первые 10 цифр общие, а дальше используют заглавные латинские буквы. Шестнадцатеричная цифра A соответствует десятеричному числу 10, шестнадцатеричная B — десятичному числу 11 и т. д. Использование этих систем объясняется тем, что переход к записи числа в любой из этих систем от его двоичной записи очень прост. Ниже приведена таблица соответствия чисел, записанных в разных системах.

Таблица 3.3. Соответствие чисел, записанных в различных системах счисления

Десятичная	Двоичная	Восьмеричная	Шестнадцатеричная
1	001	1	1
2	010	2	2
3	011	3	3
4	100	4	4
5	101	5	5
6	110	6	6

Продолжение таблицы 3.3.

Десятичная	Двоичная	Восьмеричная	Шестнадцате-ричная
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

3.5.2. Правила перевода чисел из одной системы счисления в другую

Перевод чисел из одной системы счисления в другую составляет важную часть машинной арифметики. Рассмотрим основные правила перевода.

1. Для перевода двоичного числа в десятичное необходимо его записать в виде многочлена, состоящего из произведений цифр числа и соответствующей степени числа 2, и вычислить по правилам десятичной арифметики:

$$X_2 = A_n \cdot 2^{n-1} + A_{n-1} \cdot 2^{n-2} + A_{n-2} \cdot 2^{n-3} + \ldots + A_2 \cdot 2^1 + A_1 \cdot 2^0$$

При переводе удобно пользоваться таблицей степеней двойки:

Таблица 3.4. Степени числа 2

n	0	1	2	3	4	5	6	7	8	9	10
2^n	1	2	4	8	16	32	64	128	256	512	1024

Пример. Число 11101000_2 перевести в десятичную систему счисления.

$$11101000_2 = 1\cdot 2^7 + 1\cdot 2^6 + 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0 = 232_{10}$$

2. Для перевода восьмеричного числа в десятичное необходимо его записать в виде многочлена, состоящего из произведений цифр числа и соответствующей степени числа 8, и вычислить по правилам десятичной арифметики:

$$X_8 = A_n \cdot 8^{n-1} + A_{n-1} \cdot 8^{n-2} + A_{n-2} \cdot 8^{n-3} + ... + A_2 \cdot 8^1 + A_1 \cdot 8^0$$

При переводе удобно пользоваться таблицей степеней восьмерки:

Таблица 3.5. Степени числа 8

n	0	1	2	3	4	5	6
8^n	1	8	64	512	4096	32768	262144

Пример. Число 75013_8 перевести в десятичную систему счисления.

$$75013_8 = 7\cdot 8^4 + 5\cdot 8^3 + 0\cdot 8^2 + 1\cdot 8^1 + 3\cdot 8^0 = 31243_{10}$$

3. Для перевода шестнадцатеричного числа в десятичное необходимо его записать в виде многочлена, состоящего из произведений цифр числа и соответствующей степени числа 16, и вычислить по правилам десятичной арифметики:

$$X_{16} = A_n \cdot 16^{n-1} + A_{n-1} \cdot 16^{n-2} + A_{n-2} \cdot 16^{n-3} + ... + A_2 \cdot 16^1 + A_1 \cdot 16^0$$

При переводе удобно пользоваться таблицей степеней числа 16:

Таблица 3.6. Степени числа 16

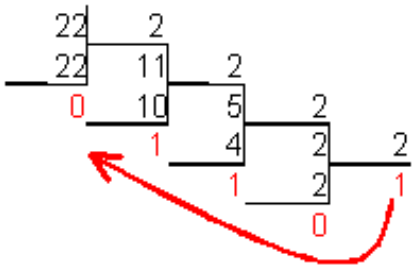
n	0	1	2	3	4	5	6
16^n	1	16	256	4096	65536	1048576	16777216

Пример. Число $FDA1_{16}$ перевести в десятичную систему счисления.

$$FDA1_{16} = 15 \cdot 163 + 13 \cdot 16^2 + 10 \cdot 16^1 + 1 \cdot 16^0 = 64929_{10}$$

4. Для перевода десятичного числа в двоичную систему его необходимо последовательно делить на 2 до тех пор, пока не останется остаток, меньший или равный 1. Число в двоичной системе записывается как последовательность последнего результата деления и остатков от деления в обратном порядке.

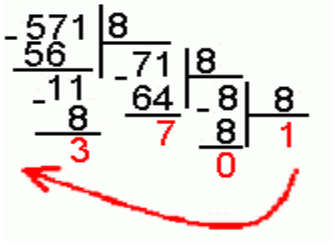
Пример. Число 22_{10} перевести в двоичную систему счисления.



$$22_{10} = 10110_2$$

5. Для перевода десятичного числа в восьмеричную систему его необходимо последовательно делить на 8 до тех пор, пока не останется остаток, меньший или равный 7. Число в восьмеричной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

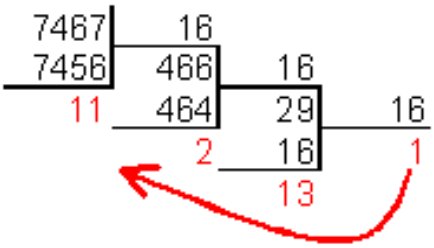
Пример. Число 571_{10} перевести в восьмеричную систему счисления.



$$571_{10} = 1073_8$$

6. Для перевода десятичного числа в шестнадцатеричную систему его необходимо последовательно делить на 16 до тех пор, пока не останется остаток, меньший или равный 15. Число в шестнадцатеричной системе записывается как последовательность цифр последнего результата деления и остатков от деления в обратном порядке.

Пример. Число 7467_2 перевести в шестнадцатеричную систему счисления.



$$7467_{10} = 1D2B_{16}$$

7. Чтобы перевести число из двоичной системы в восьмеричную, его нужно разбить на триады (тройки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую триаду нулями, и каждую триаду заменить соответствующей восьмеричной цифрой (табл. 3.1).

Пример. Число 1001011_2 перевести в восьмеричную систему счисления.

$$001\ \ 001\ \ 011_2 = 113_8$$

8. Чтобы перевести число из двоичной системы в шестнадцатеричную, его нужно разбить на тетрады (четверки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую тетраду нулями, и каждую тетраду заменить соответствующей восьмеричной цифрой (табл. 3.1).

Пример. Число 1011100011_2 перевести в шестнадцатеричную систему счисления.

$$0010\ 1110\ 0011_2 = 2E3_{16}$$

9. Для перевода восьмеричного числа в двоичное необходимо каждую цифру заменить эквивалентной ей двоичной триадой.

Пример. Число 531_8 перевести в двоичную систему счисления.

$$531_8 = 101011001_2$$

10. Для перевода шестнадцатеричного числа в двоичное необходимо каждую цифру заменить эквивалентной ей двоичной тетрадой.

Пример. Число $EE8_{16}$ перевести в двоичную систему счисления.

$$EE8_{16} = 111011101000_2$$

11. При переходе из восьмеричной системы счисления в шестнадцатеричную и обратно, необходим промежуточный перевод чисел в двоичную систему.

Пример 1. Число FEA_{16} перевести в восьмеричную систему счисления.

$$FEA_{16} = 111111101010_2$$
$$111\ 111\ 101\ 010_2 = 7752_8$$

Пример 2. Число 6653_8 перевести в шестнадцатеричную систему счисления.

$$6653_8 = 110110101011_2$$
$$1101\ 1010\ 1011_2 = DAB_{16}$$

3.5.3. Арифметические действия над целыми числами в 2-ой системе счисления.

1.Операция сложения выполняется с использованием таблицы двоичного сложения в одном разряде:

+	0	1
0	0	1
1	1	10_2

Пример.

а) $+1001_2$ б) $+1101_2$ в) $+11111_2$

1010_2 1011_2 1_2

10011_2 11000_2 100000_2

2.Операция вычитания выполняется с использованием таблицы вычитания, в которой 1 обозначается заем в старшем разряде.

–	0	1
0	0	11_2
1	1	0

Пример.

а) -101110011_2 б) -110101101_2

100011011_2 101011111_2

001011000_2 001001110_2

3.Операция умножения выполняется по обычной схеме, применяемой в десятичной с/с с последовательным умножением множимого на очередную цифру множителя.

×	0	1
0	0	0
1	0	1

Пример.

а) $\times 11001_2$ б) $\times 101_2$

1101_2 11_2

11001 101

11001 101

11001 1111_2

101000101_2

4.Операция деления выполняется по алгоритму, подобному алгоритму выполнения операции деления в 10–ой с/с.

Пример.

101000101_2 | 1101_2
1101 | 1101_2
1110
1101
1101
1101
0

100011000_2 | 1111_2
1111 | 10010_2
0010100
1111
1010_2 –остаток

3.5.4. Сложение и вычитание в восьмеричной системе счисления.

При выполнении сложения и вычитания в 8-ой с/с необходимо соблюдать следующие правила:

1) в записи результатов сложения и вычитания могут быть использованы только цифры восьмеричного алфавита;

2) десяток восьмеричной системы счисления равен 8, т.е. переполнение разряда наступает, когда результат сложения больше или равен 8.

В этом случае для записи результата надо вычесть 8, записать остаток, а к старшему разряду прибавить единицу переполнения;

3)если при вычитании приходится занимать единицу в старшем разряде, эта единица переносится в младший разряд в виде восьми единиц.

Пример

$+ 770_8$ $+ 750_8$
236_8 236_8
1226_8 512_8

3.5.5. Сложение и вычитание в шестнадцатеричной системе счисления.

При выполнении этих действий в 16–ой с/с необходимо соблюдать следующие правила:

1)при записи результатов сложения и вычитания надо использовать цифры шестнадцатеричного алфавита: цифры, обозначающие числа от 10 до 15 записываются латинскими буквами, поэтому, если результат является числом из этого промежутка, его надо записывать соответствующей латинской буквой;

2)десяток шестнадцатеричной системы счисления равен 16, т.е. переполнение разряда поступает, если результат сложения больше или равен 16, и в этом случае для записи результата надо вычесть 16, записать остаток, а к старшему разряду прибавить единицу переполнения;

3)если приходится занимать единицу в старшем разряде, эта единица переносится в младший разряд в виде шестнадцати единиц.

Примеры.

$+\ B09_{16}$	$+\ B09_{16}$
TFA_{16}	$7FA_{16}$
$1A03_{16}$	$30F_{16}$

3.5.6. Контрольные вопросы

1. Что называется системой счисления?

2. На какие два типа можно разделить все системы счисления?

3. Какие системы счисления называются непозиционными? Почему? Приведите пример такой системы счисления и записи чисел в ней?

4. Какие системы счисления применяются в вычислительной технике: позиционные или непозиционные? Почему?

5. Какие системы счисления называются позиционными?

6. Как изображается число в позиционной системе счисления?

7. Что называется основанием системы счисления?

8. Что называется разрядом в изображении числа?

9. Как можно представить целое положительное число в позиционной системе счисления?

10. Приведите пример позиционной системы счисления.

11. Опишите правила записи чисел в десятичной системе счисления:

а) какие символы образуют алфавит десятичной системы счисления?

б) что является основанием десятичной системы счисления?

в) как изменяется вес символа в записи числа в зависимости от занимаемой позиции?

12. Какие числа можно использовать в качестве основания системы счисления?

13. Какие системы счисления применяются в компьютере для представления информации?

14. Охарактеризуйте двоичную систему счисления: алфавит, основание системы счисления, запись числа.

15. Почему двоичная система счисления используется в информатике?

16. Дайте характеристику шестнадцатеричной системе счисления: алфавит, основание, запись чисел. Приведите примеры записи чисел.

17. По каким правилам выполняется сложение двух положительных целых чисел?

18. Каковы правила выполнения арифметических операций в двоичной системе счисления?

19. Для чего используется перевод чисел из одной системы счисления в другую?

20. Сформулируйте правила перевода чисел из системы счисления с основанием p в десятичную систему счисления и обратного перевода: из десятичной системы счисления в систему счисления с основанием S. Приведите примеры.

21. Как выполнить перевод чисел из двоичной СС в восьмеричную и обратный перевод? Из двоичной СС в шестнадцатеричную и обратно? Приведите примеры. Почему эти правила так просты?

23. По каким правилам выполняется перевод из восьмеричной в шестнадцатеричную СС и наоборот? Приведите примеры.

3.5.7. Задания

1. Запишите римскими цифрами числа от 1 до 100.

2. Найдите максимальное число, которое можно записать римскими цифрами.

3. Запишите в двоичной системе числа от 1 до 32.

4. Пересчитайте в мегабайты: 10240 *Кб*, 1024000 *Кб*, 10 *Гб*, 1000 *Гб*.

5. Перевести числа из 10–ой с/с в 2–ую систему счисления следующие числа: 165, 198, 541, 849, 127, 195, 289, 513, 600, 720.

6. Перевести числа из 2–ой в 10–ую систему счисления следующие числа: 110101, 100111, 1101100, 1011101, 11011101, 10010100, 111001010, 110001011

7. Произвести сложение в 2–ой системе счисления:

1) $+\ 10010011_2$ 1011011_2 2) $+\ 1011101_2$ 11101101_2 3) $+\ 10110011_2$ 1010101_2

8. Произвести вычитание в 2–ой системе счисления

1) $-\ 100001000_2$ 10110011_2 2) $-\ 110101110_2$ 10111111_2 3) $-\ 11101110_2$ 1011011_2

9. Произвести умножение в 2–ой системе счисления

1) $\times\ 100001_2$ 111111_2 2) $\times\ 100101_2$ 111011_2 3) $\times\ 111101_2$ 111101_2

10. Произвести деление в 2–ой системе счисления

1) $111010001001_2 / 111101_2$
2) $100011011100_2 / 110110_2$

3) 10000001111_2 / 111111_2

11. Переведите числа из 2–ой с/с в 8–ую ,16–ую с/с

1) 100101110_2 2) 100000111_2 3) 111001011_2
4) 1000111011_2 5) 1011001011_2 6) 110011001011_2

12. Переведите числа из 10–ой с/с в 8–ую, 16–ую с/с

1)69 2) 73 3) 113 4) 203 5) 351 6) 641

13. Переведите числа из 8–ой с/с в 10–ую с/с

1) 35_8 2) 65_8 3) 215_8 4) 327_8 5) 532_8 6) 751_8

14. Переведите числа из 16–ой с/с в 10–ую с/с

1) $D8_{16}$ 2) $1AE_{16}$ 3) $E57_{16}$ 4) $8E5_{16}$ 5) FAD_{16} 6)ADC_{16}

15. Выполните сложение 8–ых чисел

1)	2)	3)	4)	5)	6)
$+ 715_8$	$+ 524_8$	$+ 712_8$	$+ 321_8$	$+ 5731_8$	$+ 6351_8$
373_8	57_8	763_8	765_8	1376_8	737_8

16. Выполните вычитание 8–ых чисел

1)	2)	3)	4)	5)	6)
$– 137_8$	$– 436_8$	$– 705_8$	$– 538_8$	$– 7213_8$	$– 7135_8$
72_8	137_8	76_8	57_8	537_8	756_8

17. Произвести сложение 16–ых чисел

1)	2)	3)	4)	5)
$+ A13_{16}$	$+ F0B_{16}$	$+ 2EA_{16}$	$+ ABC_{16}$	$+ A2B_{16}$
$16F_{16}$	$1DA_{16}$	FCE_{16}	$C7C_{16}$	$7F2_{16}$

18. Произвести вычитание 16–ых чисел

1)	2)	3)	4)	5)
$– À17_{16}$	$– DFA_{16}$	$– FC5_{16}$	$– DE5_{16}$	$– D3C1_{16}$
$1FC_{16}$	$1AE_{16}$	AD_{16}	AF_{16}	$D1F_{16}$

4. ОСНОВЫ ЛОГИКИ

4.1. Формы мышления

Первые учения о формах и способах рассуждений возникли в странах Древнего Востока (Китай, Индия), но в основе современной логики лежат учения, созданные древнегреческими мыслителями. Основы формальной логики заложил Аристотель, который впервые отделил логические формы мышления (речи) от его содержания.

Логика то наука о формах и способах мышления.

Законы логики отражают в сознании человека свойства, связи и отношения объектов окружающего мира. Логика позволяет строить формальные модели окружающего мира, отвлекаясь от содержательной стороны.

Мышление всегда осуществляется в каких–то формах. Основными формами мышления являются понятие, высказывание и умозаключение.

Понятие. Понятие выделяет существенные признаки объекта, которые отличают его от других объектов. Объекты, объединенные понятием, образуют некоторое множество. Например, понятие «компьютер» объединяет множество электронных устройств, которые предназначены для обработки информации и обладают монитором и клавиатурой. Даже по этому короткому описанию компьютер трудно спутать с другими объектами, например с механизмами, служащими для перемещения по дорогам и хранящимися в гаражах, которые объединяются понятием «автомобиль».

Понятие — это форма мышления, фиксирующая основные, существенные признаки объекта.

Понятие имеет две стороны: содержание и объем. Содержание понятия составляет совокупность существенных признаков объекта. Чтобы раскрыть содержание понятия, следует найти признаки, необходимые и достаточные для выделения данного объекта из множества других объ-

ектов.

Например, содержание понятия «персональный компьютер» можно раскрыть следующим образом: «Персональный компьютер — это универсальное электронное устройство для автоматической обработки информации, предназначенное для одного пользователя».

Объем понятия определяется совокупностью предметов, на которую оно распространяется. Объем понятия «персональный компьютер» выражает всю совокупность (сотни миллионов) существующих в настоящее время в мире персональных компьютеров.

Высказывание. Свое понимание окружающего мира человек формулирует в форме высказываний (суждений, утверждений). Высказывание строится на основе понятий и по форме является повествовательным предложением.

Высказывания могут быть выражены с помощью не только естественных языков, но и формальных. Например, высказывание на естественном языке имеет вид «Два умножить на два равно четырем», а на формальном, математическом языке оно записывается в виде: «$2 \times 2 = 4$».

Об объектах можно судить верно или неверно, то есть высказывание может быть истинным или ложным. Истинным будет высказывание, в котором связь понятий правильно отражает свойства и отношения реальных вещей. Примером истинного высказывания может служить следующее: «Процессор является устройством обработки информации».

Ложным высказывание будет в том случае, когда оно не соответствует реальной действительности, например: «Процессор является устройством печати».

Высказывание не может быть выражено повелительным или вопросительным предложением, так как оценка их истинности или ложности невозможна.

Конечно, иногда истинность того или иного высказывания является относительной. Истинность высказываний может зависеть от взглядов

людей, от конкретных обстоятельств и так далее. Сегодня высказывание «На моем компьютере установлен самый современный процессор Pentium 4» истинно, но пройдет некоторое время, появится более мощный процессор, и данное высказывание станет ложным.

Высказывание — это форма мышления, в которой что-либо утверждается или отрицается о свойствах реальных предметов и отношениях между ними. Высказывание может быть либо истинно, либо ложно.

До сих пор мы рассматривали простые высказывания. На основании простых высказываний могут быть построены составные высказывания. Например, высказывание «Процессор является устройством обработки информации и принтер является устройством печати» является составным высказыванием, состоящим из двух простых, соединенных союзом «и».

Если истинность или ложность простых высказываний устанавливается в результате соглашения на основании здравого смысла, то истинность или ложность составных высказываний вычисляется с помощью использования алгебры высказываний.

Приведенное выше составное высказывание истинно, так как истинны входящие в него простые высказывания.

Умозаключение. Умозаключения позволяют на основе известных фактов, выраженных в форме суждений (высказываний), получать заключение, то есть новое знание. Примером умозаключений могут быть геометрические доказательства.

Например, если мы имеем суждение «Все углы треугольника равны», то мы можем путем умозаключения доказать, что в этом случае справедливо суждение «Этот треугольник равносторонний».

Умозаключение — это форма мышления, с помощью которой из одного или нескольких суждений (посылок) может быть получено новое суждение (заключение).

Посылками умозаключения по правилам формальной логики могут

только истинные суждения. Тогда, если умозаключение проводится в соответствии с правилами формальной логики, то оно будет истинным. В противном случае можно прийти к ложному умозаключению.

4.1.1. Вопросы:

1. Какие существуют основные формы мышления?
2. В чем состоит разница между содержанием и объемом понятия?
3. Может ли быть высказывание выражено в форме вопросительного предложения?
4. Как определяется истинность или ложность простого высказывания? Составного высказывания?

4.2. Алгебра высказываний

Алгебра высказываний была разработана для того, чтобы можно было определять истинность или ложность составных высказываний, не вникая в их содержание.

В алгебре высказываний суждениям (простым высказываниям) ставятся в соответствие логические переменные, обозначаемые прописными буквами латинского алфавита. Рассмотрим два простых высказывания:

A = «Два умножить на два равно четырем».

B = «Два умножить на два равно пяти».

Высказывания, как уже говорилось ранее, могут быть истинными или ложными. Истинному высказыванию соответствует значение логической переменной 1, а ложному — значение 0. В нашем случае первое высказывание истинно (A = 1), а второе ложно (B = 0).

В алгебре высказываний над высказываниями можно производить определенные логические операции, в результате которых получаются новые, составные высказывания. Для образования новых высказываний наиболее часто используются базовые логические операции, выражаемые с помощью логических связок «и», «или», «не».

4.2.1. Логическое умножение (конъюнкция)

Объединение двух (или нескольких) высказываний в одно с помощью союза «и» называется операцией логического умножения или конъюнкцией.

Составное высказывание, образованное в результате операции логического умножения (конъюнкции), истинно тогда и только тогда, когда истинны все входящие в него простые высказывания.

Так, из приведенных ниже четырех составных высказываний, образованных с помощью операции логического умножения, истинно только четвертое, так как в первых трех составных высказываниях хотя бы одно из простых высказываний ложно:

1). «$2\times2=5$ и $3\times3=10$»

2). «$2\times2=5$ и $3\times3=9$»

3). «$2\times2=4$ и $3\times3=10$»

4). «$2\times2=4$ и $3\times3=9$»

Перейдем теперь от записи высказываний на естественном языке к их записи на формальном языке алгебры высказываний (алгебры логики). В ней операцию логического умножения (конъюнкцию) принято обозначать значком «&» либо «$\wedge$». Образуем составное высказывание F, которое получится в результате конъюнкции двух простых высказываний:

$$F = A \ \& \ B.$$

С точки зрения алгебры высказываний записана формула функции логического умножения, аргументами которой являются логические переменные A и B, которые могут принимать значения «истина» (1) и «ложь» (0).

Сама функция логического умножения F также может принимать лишь два значения «истина» (1) и «ложь» (0). Значение логической функции можно определить с помощью таблицы истинности данной

функции, которая показывает, какие значения принимает логическая функция при всех возможных наборах ее аргументов (табл. 4.1).

Таблица 4.1. Таблица истинности функции логического умножения

A	***B***	***F = A & B***
0	0	0
0	1	0
1	0	0
1	1	1

По таблице истинности легко определить истинность составного высказывания, образованного с помощью операции логического умножения. Рассмотрим, например, составное высказывание «$2\times2=4$ и $3\times3=10$». Первое простое высказывание истинно (A = 1), а второе высказывание ложно (B = 0), по таблице определяем, что логическая функция принимает значение ложь (F = 0), то есть данное составное высказывание ложно.

4.2.2. Логическое сложение (дизъюнкция)

Объединение двух (или нескольких) высказываний с помощью союза «или» называется операцией логического сложения или дизъюнкцией.

Составное высказывание, образованное в результате логического сложения (дизъюнкции), истинно тогда, когда истинно хотя бы одно из входящих в него простых высказываний.

Так, из приведенных ниже четырех составных высказываний, образованных с помощью операции логического сложения, ложно только первое, так как в последних трех составных высказываниях хотя бы одно из простых высказываний истинно:

1). «$2\times2=5$ или $3\times3=10$»

2). «$2\times2=5$ или $3\times3=9$»

3). «$2\times2=4$ или $3\times3=10$»

4). «$2\times2=4$ или $3\times3=9$»

Запишем теперь операцию логического сложения на формальном языке алгебры логики. Операцию логического сложения (дизъюнкцию) принято обозначать либо значком «$\vee$», либо знаком сложения «+». Образуем составное высказывание F, которое получится в результате дизъюнкции двух простых высказываний:

$$F = A \vee B$$

С точки зрения алгебры высказываний записана формула функции логического сложения, аргументами которой являются логические переменные A и **B**. Значение логической функции можно определить с помощью таблицы истинности данной функции, которая показывает, какие значения принимает логическая функция при всех возможных наборах ее аргументов (табл. 4.2).

Таблица 4.2. Таблица истинности функции логического сложения

A	***B***	***F = A ∨ B***
0	0	0
0	1	1
1	0	1
1	1	1

По таблице истинности легко определить истинность составного высказывания, образованного с помощью операции логического сложения. Рассмотрим, например, составное высказывание «$2\times2=4$ или $3\times3=10$». Первое простое высказывание истинно (A = 1), а второе высказывание ложно (B = 0), по таблице определяем, что логи-

ческая функция принимает значение истина (F = 1), то есть данное составное высказывание истинно.

4.2.3. Логическое отрицание (инверсия)

Присоединение частицы «не» к высказыванию называется операцией логического отрицания или инверсией.

Логическое отрицание (инверсия) делает истинное высказывание ложным и, наоборот, ложное — истинным.

Пусть A = «Два умножить на два равно четырем» — истинное высказывание, тогда высказывание F = «Два умножить на два не равно четырем», образованное с помощью операции логического отрицания, — ложно.

Операцию логического отрицания (инверсию) над логическим высказыванием A в алгебре логики принято обозначать A. Образуем высказывание F, являющееся логическим отрицанием A:

$$F = \overline{A}$$

Истинность такого высказывания задается таблицей истинности функции логического отрицания (табл. 4.3).

Таблица 4.3. Таблица истинности функции логического отрицания

A	$F = \overline{A}$
0	1
1	0

Истинность высказывания, образованного с помощью операции логического отрицания, можно легко определить с помощью таблицы истинности. Например, высказывание «Два умножить на два не равно четырем» ложно (A = 0), а полученное из него в результате логического отрицания высказывание «Два умножить на два равно четырем» истинно (F = 1).

4.2.4. Логические выражения и таблицы истинности

Каждое составное высказывание можно выразить в виде формулы (логического выражения), в которую входят логические переменные, обозначающие высказывания, и знаки логических операций, обозначающие логические функции.

Для записи составного высказывания в виде логического выражения на формальном языке (языке алгебры логики) в составном высказывании нужно выделить простые высказывания и логические связи между ними.

Запишем в форме логического выражения составное высказывание «(2×2=5 или 2×2=4) и (2×2≠5 или 2×2≠4)». Проанализируем составное высказывание. Оно содержит два простых высказывания:

A = «2×2=5 » — ложно (0),

B = «2×2=4 » — истинно(1).

Тогда составное высказывание можно записать в следующей форме:

«(A или B) и ($\overline{A}$ или $\overline{B}$)».

Теперь необходимо записать высказывание в форме логического выражения с учетом последовательности выполнения логических операций. При выполнении логических операций определен следующий порядок их выполнения: инверсия, конъюнкция, дизъюнкция. Для изменения указанного порядка могут использоваться скобки:

$$F = (A \vee B) \,\&\, (\overline{A} \vee \overline{B}).$$

Истинность или ложность составных высказываний можно определять чисто формально, руководствуясь законами алгебры высказываний, не обращаясь к смысловому содержанию высказываний.

Подставим в логическое выражение значения логических переменных и, используя таблицы истинности базовых логических операций, получим значение логической функции:

$$F = (A \vee B) \,\&\, (\overline{A} \vee \overline{B}) = (0 \vee 1) \,\&\, (1 \vee 0) = 1 \,\&\, 1 = 1$$

Для каждого составного высказывания (логического выражения) можно построить таблицу истинности, которая определяет его истинность или ложность при всех возможных комбинациях исходных значений простых высказываний (логических переменных).

При построении таблиц истинности целесообразно руководствоваться определенной последовательностью действий.

Во–первых, необходимо определить количество строк в таблице истинности. Оно равно количеству возможных комбинаций значений логических переменных, входящих в логическое выражение. Если количество логических переменных равно n, то:

$$\text{количество строк} = 2^n. \qquad (4.1)$$

Для рассмотренного примера логическая функция $F = (A \vee B) \,\&\, (\overline{A} \vee \overline{B})$ имеет 2 переменные и, следовательно, количество строк в таблице истинности должно быть равно 4.

Во–вторых, необходимо определить количество столбцов в таблице истинности, которое равно количеству логических переменных плюс количество логических операций.

Функция $F = (A \vee B) \,\&\, (\overline{A} \vee \overline{B})$ содержит две переменные и пять логических операций, то есть количество столбцов таблицы истинности равно семи.

В–третьих, необходимо построить таблицу истинности с указанным количеством строк и столбцов, обозначить столбцы и внести в таблицу возможные наборы значений исходных логических переменных.

В–четвертых, необходимо заполнить таблицу истинности по столбцам, выполняя базовые логические операции в необходимой последовательности и в соответствии с их таблицами истинности (табл. 4.4). Теперь мы можем определить значение логической функции для любого набора значений логических переменных.

Таблица 4.4. Таблица истинности логической функции $F = (A \vee B) \& (\overline{A} \vee \overline{B})$

A	B	$A \vee B$	$\overline{A}$	$\overline{B}$	$\overline{A} \vee \overline{B}$	$(A \vee B) \& (\overline{A} \vee \overline{B})$
0	0	0	1	1	1	0
0	1	1	1	0	1	1
1	0	1	0	1	1	1
1	1	1	0	0	0	0

Равносильные логические выражения. Логические выражения, у которых последние столбцы таблиц истинности совпадают, называются равносильными. Для обозначения равносильных логических выражений используется знак «=».

Докажем, что логические выражения $\overline{A} \& \overline{B}$ и $\overline{A \vee B}$ равносильны. Построим сначала таблицу истинности логического выражения $\overline{A} \& \overline{B}$ (табл. 4.5).

Таблица 4.5. Таблица истинности логического выражения $\overline{A} \& \overline{B}$

A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \& \overline{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	0

Теперь построим таблицу истинности логического выражения $\overline{A \vee B}$(табл. 4.6).

Значения в последних столбцах таблиц истинности совпадают, следовательно, логические выражения равносильны:

$$\overline{A} \& \overline{B} = \overline{A \vee B}$$

Таблица 4.6. Таблица истинности логического выражения $\overline{A \vee B}$

A	B	$A \vee B$	$\overline{A \vee B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

4.2.4.1. Задания

1. Записать составное высказывание «$(2\times2=4$ и $3\times3=9)$ или $(2\times2\neq4$ и $3\times3\neq9)$» в форме логического выражения. Построить таблицу истинности.

2. Доказать, используя таблицы истинности, что логические выражения $\overline{\overline{A} \vee \overline{B}}$ и $A\&B$ равносильны.

4.2.5. Логические функции

Любое составное высказывание можно рассматривать как логическую функцию $F(X_1, X_2, \ldots, X_n)$, аргументами которой являются логические переменные $X_1, X_2, \ldots, X_n$ (простые высказывания). Сама функция и аргументы могут принимать только два различных значения: «истина» (1) и «ложь» (0).

Выше были рассмотрены функции двух аргументов: логическое умножение $F(A,B) = A \,\&\, B$, логическое сложение $F(A,B) = A \vee B$, а также логическое отрицание $F(A) = \overline{A}$, в котором значение второго аргумента можно считать равным нулю.

Каждая логическая функция двух аргументов имеет четыре возможных набора значений аргументов. По формуле (4.1) мы можем определить, какое количество различных логических функций двух аргументов может существовать:

$$N = 2^4 = 16.$$

Таким образом, существует 16 различных логических функций двух аргументов, каждая из которых задается своей таблицей истинности (табл. 4.7).

Таблица 4.7. Таблицы истинности логических функций двух аргументов

Аргументы		**Логические функции**															
A	***B***	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Легко заметить, что здесь логическая функция F_2 является функцией логического умножения, F_8 — функцией логического сложения, F_{13} — функцией логического отрицания для аргумента A и F_{11} — функцией логического отрицания для аргумента B.

Логическое следование (импликация). Логическое следование (импликация) образуется соединением двух высказываний в одно с помощью оборота речи «если..., то...».

Логическая операция импликации «если A, то B», обозначается $A \rightarrow B$ и выражается с помощью логической функции F_{14}, которая задается соответствующей таблицей истинности (табл. 4.8).

Составное высказывание, образованное с помощью операции логического следования (импликации), ложно тогда и только тогда, когда из истинной предпосылки (первого высказывания) следует ложный вывод (второе высказывание).

Таблица 4.8. Таблица истинности логической функции «импликация»

A	B	$F_{14} = A \to B$
0	0	1
0	1	1
1	0	0
1	1	1

Например, высказывание «Если число делится на 10, то оно делится на 5» истинно, так как истинны и первое высказывание (предпосылка), и второе высказывание (вывод). Высказывание «Если число делится на 10, то оно делится на 3» ложно, так как из истинной предпосылки делается ложный вывод. Однако операция логического следования несколько отличается от обычного понимания слова «следует». Если первое высказывание (предпосылка) ложно, то вне зависимости от истинности или ложности второго высказывания (вывода) составное высказывание истинно. Это можно понимать таким образом, что из неверной предпосылки может следовать что угодно. В алгебре высказываний все логические функции могут быть сведены путем логических преобразований к трем базовым: логическому умножению, логическому сложению и логическому отрицанию. Докажем методом сравнения таблиц истинности (табл. 4.8 и 4.9), что операция импликации $A \to B$ равносильна логическому выражению $\overline{A} \vee B$.

Таблица 4.9. Таблица истинности логического выражения $\overline{A} \vee B$

A	B	$\overline{A}$	$\overline{A} \vee B$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

Таблицы истинности совпадают, что и требовалось доказать.

Логическое равенство (эквивалентность). Логическое равенство (эквивалентность) образуется соединением двух высказываний в одно с помощью оборота речи «... тогда и только тогда, когда ...».

Логическая операция эквивалентности «*A* тогда и только тогда, когда *B*» обозначается *A*~*B* и выражается с помощью логической функции F_{10}, которая задается соответствующей таблицей истинности (табл. 4.10).

Таблица 4.10. Таблица истинности логической функции эквивалентности

A	*B*	$F_{10} = A \sim B$
0	0	1
0	1	0
1	0	0
1	1	1

Составное высказывание, образованное с помощью логической операции эквивалентности истинно тогда и только тогда, когда оба высказывания одновременно либо ложны, либо истинны.

Рассмотрим, например, два высказывания: *A* = «Компьютер может производить вычисления» и *B* = «Компьютер включен». Составное высказывание, полученное с помощью операции эквивалентности, истинно, когда оба высказывания либо истинны, либо ложны:

«Компьютер может производить вычисления тогда и только тогда, когда компьютер включен».

«Компьютер не может производить вычисления тогда и только тогда, когда компьютер не включен».

Составное высказывание, полученное с помощью операции эквивалентности, ложно, когда одно высказывание истинно, а другое — ложно:

«Компьютер может производить вычисления тогда и только тогда, когда компьютер не включен».

«Компьютер не может производить вычисления тогда и только тогда, когда компьютер включен».

4.2.5.1. Вопросы и задания.

1. Какое количество логических функций двух аргументов существует и почему?

2. Какие логические функции двух аргументов имеют свои названия?

3. Доказать, используя таблицы истинности, что операция эквивалентности $A \sim B$ равносильна логическому выражению: $(A \vee \overline{B}) \& (\overline{A} \vee B)$.

4.3. Логические законы и правила преобразования логических выражений

Законы логики отражают наиболее важные закономерности логического мышления. В алгебре высказываний законы логики записываются в виде формул, которые позволяют проводить эквивалентные преобразования логических выражений.

Закон тождества. Всякое высказывание тождественно самому себе: $A = A$.

Закон непротиворечия. Высказывание не может быть одновременно истинным и ложным. Если высказывание A истинно, то его отрицание не A должно быть ложным. Следовательно, логическое произведение высказывания и его отрицания должно быть ложно: $A \& \overline{A} = 0$

Закон исключенного третьего. Высказывание может быть либо истинным, либо ложным, третьего не дано. Это означает, что результат логического сложения высказывания и его отрицания всегда принимает значение «истина»: $A + \overline{A} = 1$

Закон двойного отрицания. Если дважды отрицать некоторое высказывание, то в результате мы получим исходное высказывание: $\overline{\overline{A}} = A$

Законы де Моргана.

$$\overline{A \vee B} = \overline{A} \,\&\, \overline{B}$$

$$\overline{A \,\&\, B} = \overline{A} \vee \overline{B}$$

Для выполнения преобразований логических выражений важное значение имеют законы алгебраических преобразований. Многие из них имеют аналоги в обычной алгебре.

Закон коммутативности. В обычной алгебре слагаемые и множители можно менять местами. В алгебре высказываний можно менять местами логические переменные при операциях логического умножения и логического сложения:

Логическое умножение	Логическое сложение
$A \,\&\, B = B \,\&\, A$	$A \vee B = B \vee A$

Закон ассоциативности. Если в логическом выражении используются только операция логического умножения или только операция логического сложения, то можно пренебрегать скобками или произвольно их расставлять:

Логическое умножение	Логическое сложение
$(A \,\&\, B) \,\&\, C = A \,\&\, (B \,\&\, C)$	$(A \vee B) \,\&\, C = A \vee (B \vee C)$

Закон дистрибутивности. В отличие от обычной алгебры, где за скобки можно выносить только общие множители, в алгебре высказываний можно выносить за скобки как общие множители, так и общие слагаемые:

Дистрибутивность умножения относительно сложения	Дистрибутивность сложения относительно умножения
$(A \,\&\, B) \vee (A \,\&\, C) = A \,\&\, (B \vee C)$	$A \vee B) \,\&\, (A \vee C) = A \vee (B \,\&\, C)$

Рассмотрим в качестве примера применения законов логики преобразование логического выражения. Пусть нам необходимо упростить логическое выражение:

$$(A \,\&\, B) \vee (A \,\&\, \overline{B}).$$

Воспользуемся законом дистрибутивности и вынесем за скобки A:

$$(A \,\&\, B) \vee (A \,\&\, B) = A \,\&\, (B \vee \overline{B}).$$

По закону исключенного третьего $B \vee \overline{B} .= 1$, следовательно:

$$(A \,\&\, B) \vee (A \,\&\, \overline{B}). = A \,\&\, 1 = A.$$

4.3.1. Задания

Доказать справедливость первого $\overline{A \vee B} = \overline{A} \,\&\, \overline{B}$ и второго $\overline{A \,\&\, B} = \overline{A} \vee \overline{B}$ законов де Моргана, используя таблицы истинности.

3.6. Упростить логические выражения:

а) $(A \vee \overline{A}) \,\&\, B$;

б) $A \,\&\, (A \vee B) \,\&\, (B \vee \overline{B})$.

4.4. Решение логических задач

Логические задачи обычно формулируются на естественном языке. В первую очередь их необходимо формализовать, то есть записать на языке алгебры высказываний. Полученные логические выражения необходимо упростить и проанализировать. Для этого иногда бывает необходимо построить таблицу истинности полученного логического выражения.

Пример 1. Преобразование логических выражений.

Условие задачи. В школе-новостройке в каждой из двух аудиторий

может находиться либо кабинет информатики, либо кабинет физики. На дверях аудиторий повесили шутливые таблички. На первой повесили табличку «По крайней мере, в одной из этих аудиторий размещается кабинет информатики», а на второй аудитории — табличку с надписью «Кабинет физики находится в другой аудитории». Проверяющему, который пришел в школу, известно только, что надписи на табличках либо обе истинны, либо обе ложны. Помогите проверяющему найти кабинет информатики.

Решение задачи. Переведем условие задачи на язык логики высказываний. Так как в каждой из аудиторий может находиться кабинет информатики, то пусть:

A = «В первой аудитории находится кабинет информатики»;

B = «Во второй аудитории находится кабинет информатики».

Отрицания этих высказываний:

$\overline{A}$ = «В первой аудитории находится кабинет физики»;

$\overline{B}$ = «Во второй аудитории находится кабинет физики».

Высказывание, содержащееся на табличке на двери первой аудитории, соответствует логическому выражению:

$$X = A \vee B.$$

Высказывание, содержащееся на табличке на двери второй аудитории, соответствует логическому выражению:

$$Y = \overline{A}.$$

Содержащееся в условии задачи утверждение о том, что надписи на табличках либо одновременно истинные, либо одновременно ложные в соответствии с законом исключенного третьего записывается следующим образом:

$$(X \,\&\, Y) \vee (\overline{X} \,\&\, \overline{Y}) = 1$$

Подставим вместо X и Y соответствующие формулы:

$$(X \,\&\, Y) \vee (\overline{X} \,\&\, \overline{Y}) = ((A \vee B) \,\&\, \overline{A}) \vee (\overline{(A \vee B)} \,\&\, \overline{\overline{A}})..$$

Упростим сначала первое слагаемое. В соответствии с законом дистрибутивности умножения относительно сложения:

$(A \vee B) \& \overline{A} = A \& \overline{A} \vee B \& \overline{A}$.

В соответствии с законом непротиворечия:

$A \& \overline{A} \vee B \& \overline{A} = 0 \vee B \& \overline{A}$

Упростим теперь второе слагаемое. В соответствии с первым законом де Моргана и законом двойного отрицания:

$\overline{(A \vee B)} \& \overline{\overline{A}} = \overline{A} \& \overline{B} \& A = \overline{A} \& A \& \overline{B}.$

В соответствии с законом непротиворечия:

$\overline{A} \& A \& \overline{B} = 0 \& \overline{B} = 0.$

В результате получаем:

$(0 \vee B \& \overline{A}) \vee 0 = B \& \overline{A}$

Полученное логическое выражение оказалось простым и поэтому его можно проанализировать без построения таблицы истинности Для того чтобы выполнялось равенство $B \& \overline{A} = 1$, B и $\overline{A}$ должны быть равны 1, то есть соответствующие им высказывания истинны.

Ответ: В первой аудитории находится кабинет физики, а во второй — кабинет информатики.

Пример 2. Табличный способ решения

Условие задачи. Покупатель в каждом из магазинов *A*, *B*, *C*, *D* сделал по одной покупке и приобрел джойстик, дискеты, бумагу и картридж, известно, что:

- джойстик и картридж покупатель купил не в «*A*»
- в «*C*» зашел, когда уже купил дискеты и бумагу
- в «*D*» не было ни картриджа, ни дискет
- в «*B*» покупатель приехал, когда джойстик уже был куплен, а из «*D*» уходил еще без джойстика.

Вопрос:

Что было куплено в магазинах *A*, *B*, *C*, *D*?

Решение задачи:

Составим таблицу:

	A	*B*	*C*	*D*
джойстик				
дискета				
бумага				
картридж				

и заполним ее.

1. Так как известно, что джойстик и картридж куплен не в «*A*», то ставим «–» в соответствующих ячейках.

	A	B	C	D
джойстик	–			
дискета				
бумага				
картридж	–			

3.В магазин «*C*» покупатель зашел, когда уже купил дискеты и бумагу. Это означает, что в «*C*» он их не покупал! Получается:

	A	B	C	D
джойстик	–			
дискета			–	
бумага			–	
картридж	–			

4.По условию в «*D*» не было ни картриджа, ни дискет. Значит, ставим «–» в соответствующие клетки:

	A	B	C	D
джойстик	–			
дискета			–	–
бумага			–	
картридж	–			–

5.И, наконец, в магазин «*B*» покупатель приехал, когда джойстик уже был куплен, значит, в «*B*» он его не покупал (ставим «–»). *A* из «*D*» уходил еще без джойстика, следовательно, в «*D*» он его не покупал, т.е. там тоже появится «–».

	A	B	C	D
джойстик	–	–	+	–
дискета			–	–
бумага			–	
картридж	–			–

Но, раз джойстик не приобретен ни в *A*, ни в *B*, ни в *D*, значит, он куплен в *C*!(появляется «+» в таблице).

5. Так как везде он приобретал что–нибудь одно, то дальнейшее решение содержит следующие рассуждения:

– в «*C*» больше не может быть «+», следовательно, ставим последний в данном столбце «–»,

	A	B	C	D
джойстик	–	–	+	–
дискета			–	–
бумага			–	
картридж	–		–	–

– в «*D*» больше не может быть «–», следовательно, ставим недостающий «+»,

	A	B	C	D
джойстик	–	–	+	–
дискета			–	–
бумага			–	+
картридж	–		–	–

– анализ последней строки позволяет сделать вывод, картридж может быть куплен только в «B», ставим в соответствующую ячейку «+»

	A	B	C	D
джойстик	–	–	+	–
дискета		–	–	–
бумага			–	+
картридж	–	+	–	–

– дальнейший анализ полученной таблицы показывает, что бумага уже куплена, значит, смело можно ставить минусы против A и B. Кроме того, сделана уже покупка в B, значит, и в столбце B тоже ставим минус.

	A	B	C	D
джойстик	–	–	+	–
дискета		–	–	–
бумага	–	–	–	+
картридж	–	+	–	–

– осталась незаполненной одна единственная ячейка, в которую заносим последний символ «+» нашего решения.

	A	B	C	D
джойстик	–	–	+	–
дискета	+	–	–	–
бумага	–	–	–	+
картридж	–	+	–	–

4.4.1. Задания.

1. В процессе составления расписания уроков учителя высказали свои пожелания. Учитель математики высказал пожелание проводить первый или второй урок, учитель информатики — первый и третий, а

учитель физики — второй или третий урок. Сколько существует возможных вариантов расписания и каковы они?

2. Трое друзей, болельщиков автогонок "Формула-1", спорили о результатах предстоящего этапа гонок.

— Вот увидишь, Шумахер не придет первым, — сказал Джон. Первым будет Хилл.

— Да нет же, победителем будет, как всегда, Шумахер, — воскликнул Ник. — А об Алези и говорить нечего, ему не быть первым.

Питер, к которому обратился Ник, возмутился:

— Хиллу не видать первого места, а вот Алези пилотирует самую мощную машину.

По завершении этапа гонок оказалось, что каждое из двух предположений двоих друзей подтвердилось, а оба предположения третьего из друзей оказались неверны. Кто выиграл этап гонки?

3. Некий любитель приключений отправился в кругосветное путешествие на яхте, оснащённой бортовым компьютером. Его предупредили, что чаще всего выходят из строя три узла компьютера — ***a***, ***b***, ***c***, и дали необходимые детали для замены. Выяснить, какой именно узел надо заменить, он может по сигнальным лампочкам на контрольной панели. Лампочек тоже ровно три: ***x***, ***y*** и ***z***.

Инструкция по выявлению неисправных узлов такова:

1. если неисправен хотя бы один из узлов компьютера, то горит по крайней мере одна из лампочек ***x***, ***y***, ***z***;

2. если неисправен узел ***a***, но исправен узел ***c***, то загорается лампочка ***y***;

3. если неисправен узел ***c***, но исправен узел ***b***, загорается лампочка ***y***, но не загорается лампочка ***x***;

4. если неисправен узел ***b***, но исправен узел ***c***, то загораются лампочки ***x*** и ***y*** или не загорается лампочка ***x***;

5. если горит лампочка ***x*** и при этом либо неисправен узел ***a***, либо все три узла ***a, b, c*** исправны, то горит и лампочка ***y***.

В пути компьютер сломался. На контрольной панели загорелась лампочка ***x***. Тщательно изучив инструкцию, путешественник починил компьютер. Но с этого момента и до конца плавания его не оставляла тревога. Он понял, что инструкция несовершенна, и есть случаи, когда она ему не поможет.

Какие узлы заменил путешественник? Какие изъяны он обнаружил в инструкции?

4. В симфонический оркестр приняли на работу трёх музыкантов: Брауна, Смита и Вессона, умеющих играть на скрипке, флейте, альте, кларнете, гобое и трубе.

Известно, что:

1. Смит самый высокий;

2. играющий на скрипке меньше ростом играющего на флейте;

3. играющие на скрипке и флейте и Браун любят пиццу;

4. когда между альтистом и трубачом возникает ссора, Смит мирит их;

5. Браун не умеет играть ни на трубе, ни на гобое.

На каких инструментах играет каждый из музыкантов, если каждый владеет двумя инструментами?

5. Три одноклассника — Влад, Тимур и Юра, встретились спустя 10 лет после окончания школы. Выяснилось, что один из них стал врачом, другой физиком, а третий юристом. Один полюбил туризм, другой бег, страсть третьего — регби.

Юра сказал, что на туризм ему не хватает времени, хотя его сестра — единственный врач в семье, заядлый турист. Врач сказал, что он разделяет увлечение коллеги.

Забавно, но у двоих из друзей в названиях их профессий и увлечений не встречается ни одна буква их имен.

Определите, кто чем любит заниматься в свободное время и у кого какая профессия.

6. Три дочери писательницы Дорис Кей — Джуди, Айрис и Линда, тоже очень талантливы. Они приобрели известность в разных видах искусств — пении, балете и кино. Все они живут в разных городах, поэтому Дорис часто звонит им в Париж, Рим и Чикаго.

Известно, что:

1. Джуди живет не в Париже, а Линда — не в Риме;

2. парижанка не снимается в кино;

3. та, кто живет в Риме, певица;

4. Линда равнодушна к балету.

Где живет Айрис, и какова ее профессия?

7. Вадим, Сергей и Михаил изучают различные иностранные языки: китайский, японский и арабский. На вопрос, какой язык изучает каждый из них, один ответил: "Вадим изучает китайский, Сергей не изучает китайский, а Михаил не изучает арабский". Впоследствии выяснилось, что в этом ответе только одно утверждение верно, а два других ложны. Какой язык изучает каждый из молодых людей?

8. В поездке пятеро друзей — Антон, Борис, Вадим, Дима и Гриша, знакомились с попутчицей. Они предложили ей отгадать их фамилии, причём каждый из них высказал одно истинное и одно ложное утверждение:

Дима сказал: "Моя фамилия — Мишин, а фамилия Бориса — Хохлов". Антон сказал: "Мишин — это моя фамилия, а фамилия Вадима — Белкин". Борис сказал: "Фамилия Вадима — Тихонов, а моя фамилия — Мишин". Вадим сказал: "Моя фамилия — Белкин, а фамилия Гриши — Чехов". Гриша сказал: "Да, моя фамилия Чехов, а фамилия Антона — Тихонов".

Какую фамилию носит каждый из друзей?

9. Министры иностранных дел России, США и Китая обсудили за

закрытыми дверями проекты соглашения о полном разоружении, представленные каждой из стран. Отвечая затем на вопрос журналистов: "Чей именно проект был принят?", министры дали такие ответы:

Россия — "Проект не наш, проект не США";

США — "Проект не России, проект Китая";

Китай — "Проект не наш, проект России".

Один из них (самый откровенный) оба раза говорил правду; второй (самый скрытный) оба раза говорил неправду, третий (осторожный) один раз сказал правду, а другой раз — неправду.

Определите, представителями каких стран являются откровенный, скрытный и осторожный министры.

5. ПРОГРАММИРОВАНИЕ

5.1. Алгоритм, его характеристики и свойства

Понятие алгоритма является основным при составлении любого вида программ для вычислительной машины.

Алгоритм — описание последовательности действий для решения задачи.

Слово алгоритм происходит от латинской формы написания имени выдающегося узбекского математика IX века Аль–Харезми. Он впервые сформулировал правила выполнения арифметических действий с использованием арабских цифр.

Для примера рассмотрим изложенный Евклидом процесс нахождения наибольшего общего делителя двух чисел. Даны два целых положительных числа n и m. Требуется найти их наибольший общий делитель, т.е. наибольшее положительное число, на которое делится без остатка как n так и m.

Суть алгоритма при условии, что $m > n$ можно представить следующими тремя этапами [4].

1. Нахождение остатка. Разделим m на n. Пусть остаток равен r:

$$0 \le r < n$$

2. Проверка остатка на равенство нулю. Если $r = 0$, вычисление заканчивается, n — искомое число.

3. Замена $m \leftarrow n,\ n \leftarrow r$. Повторить этап 1.

Поясним на примере:

Даны $m = 125,\ n = 75$. Найти наибольший общий делитель.

1) 125:75=1. Целая часть от деления числа 125 на число 75 равна 1. Остаток от деления равен 50.

2) Остаток от деления не равен 0.

3) Проводим замену переменных: $m = 75,\ n = 50.$

Повторяем процесс.

1) 75:50=1, $r = 25$

2) Остаток от деления не равен 0.

3) $m = 50,\ n = 25,\ n \neq 0$

Повторяем процесс.

1) 50:25=2

2) Остаток равен 0, поэтому $n = 25$ является наибольшим общим делителем.

Алгоритм имеет две характеристики.

1. ***Конечность, или результативность***. Алгоритм приводит к получению результата за конечное число шагов.

2. ***Однозначность, или определенность***. При одинаковых входных данных алгоритм выдает одинаковый результат.

Алгоритм также обладает следующими свойствами.

1. ***Массовость, или универсальность***. Алгоритм выдает результат прилюбых однотипных входных данных.

2. ***Модульность, или дискретность***. Алгоритм можно представить в виде последовательности более элементарных алгоритмов.

3. ***Эффективность***. Алгоритм должен быть эффективным, т.е. результат должен быть получен наименьшим числом наиболее простых операций.

Выделяют три крупных класса алгоритмов:

- вычислительные алгоритмы, работающие со сравнительно простыми видами данных, такими как числа и матрицы, хотя сам процесс вычисления может быть долгим и сложным;

- информационные алгоритмы, представляющие собой набор сравнительно простых процедур, работающих с большими объемами информации (алгоритмы баз данных);
- управляющие алгоритмы, генерирующие различные управляющие воздействия на основе данных, полученных от внешних процессов, которыми алгоритмы управляют.

5.2. Создание алгоритмов

Основной метод создания алгоритмов — ***программирование «сверху вниз» или пошаговая детализация***. Он заключается в разбиении исходной задачи на последовательность нескольких меньших подзадач. Эти подзадачи, в свою очередь, тоже распадаются на подзадачи и т. д. до тех пор, пока не останутся только элементарные алгоритмы.

Например, чтобы написать число 512, сначала пишут цифру 5, затем 1 и, наконец, 2. При этом цифры рисуют, последовательно прорисовывая линии, из которых они состоят. Принтер напечатает это число точками.

5.3. Модульность алгоритмов, головная программа

При программировании сверху вниз алгоритмы и данные делятся на относительно независимые части, называемые ***модулями***. Некоторые из модулей являются стандартными и поставляются в составе языков программирования, например, вычисление элементарных математических функций квадратный корень, логарифм, синус и т. д. Но главные модули все равно приходится проектировать программистам.

Таким образом, алгоритм является ***деревом модулей***: одни модули вызывают другие модули, начиная с самого верхнего первого модуля, называемого ***корневым модулем, или головной программой*** (рис. 5.1).

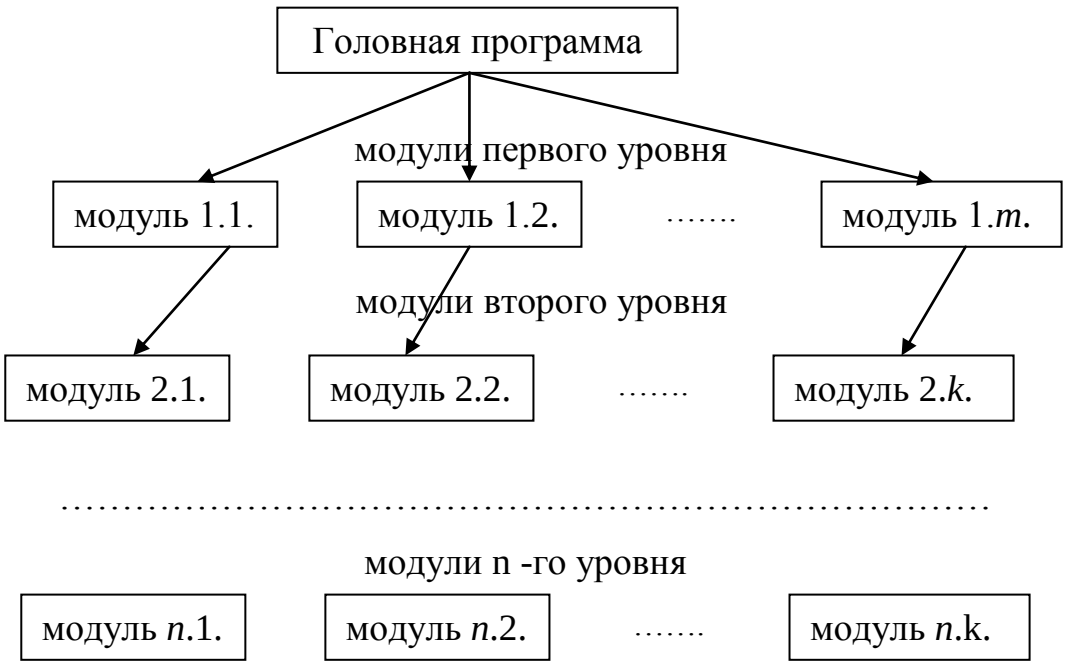


Рис. 5.1 Дерево модулей

Реализация сложных задач немыслима без применения модульного подхода проектирования программ. Каждый модуль является независимой программной единицей, реализующей законченную подзадачу. С точки зрения других модулей текущий модуль рассматривается как «черный ящик».

Принцип черного ящика означает, что при включении реализованного модуля в программу, программист может не знать, как этот модуль выполняет свою функцию, какие алгоритмы скрыты у него внутри. Для него важно следующее:

1) какова функция модуля, т. е. какую задачу он решает;

2) как использовать модуль, т.е. каковы входные и выходные данных модуля.

Таким образом, правильное проектирование алгоритмов позволяет абстрагироваться от внутренней структуры модулей и рассматривать при сборке полного алгоритма только функции модулей.

5.4. Технология программирования

5.4.1. Структурное программирование

Впервые основные идеи структурного программирования были высказаны Эдсгером Дейкстром в 1965 году и позже опубликованы в его работе [3]. Основная задача, которую Э. Дейкстра решал, разрабатывая идеи структурного программирования, была задача доказательства правильности программы. Его внимание было сосредоточено на вопросе, «какими должны быть структуры программ, чтобы без чрезмерных усилий мы могли находить доказательство их правильности».

Структурное программирование представляет собой некоторые принципы написания программ в соответствии со строгими правилами дисциплиной и имеет целью облегчить процесс тестирования, повысить производительность труда программистов, улучшить ясность и читабельность программы, а также повысить ее эффективность.

Структурное программирование можно толковать как «проектирование, написание и тестирование программы в соответствии с заранее определенной дисциплиной» [10].

Теоретической основой структурного программирования принято считать принципы, изложенные в классической работе Бома и Джакопини [12]. Эта работа в оригинале на итальянском языке была опубликована в 1965 г., а в английском переводе — в 1966 г.

Структурное программирование является технологией простого и прозрачного создания алгоритмов. Это единственный способ строить алгоритмы быстро и в последующем легко вносить в них изменения.

В соответствии с так называемой «структурной» теоремой, сформулированной и доказанной в этой работе, всякая программа может быть построена с использованием только трех основных типов блоков [4].

Кроме того, теорема структурного программирования утверждает, что любой алгоритм можно преобразовать к структурному виду.

Тремя элементарными структурными алгоритмами являются:

1. Следование, или цепочка, или составная инструкция.

2. Выбор, или ветвление, или условная инструкция.

3. Цикл, или возврат, или циклическая инструкция.

5.4.2. Объектно–ориентированное программирование

Компьютер = аппаратура + программы, а программа = алгоритм + данные. После открытия структурного проектирования алгоритмов стали разбираться с программированием данных.

Объектно–ориентированное программирование (ООП) организует данные и алгоритмы, обрабатываемые программой. При этом программист создает формы данных и алгоритмы, соответствующие основным характеристикам решаемой проблемы. Модели данных и алгоритмы, их обрабатывающие, называются ***классами***, а ***объекты*** — это конкретные их представители, используемые в программе.

Из общих объектов создаются другие, более специализированные. Механизм создания таких подобъектов называется ***наследованием***. В итоге данные программы представляют из себя ***объектную модель*** — дерево объектов, начиная с самого верхнего наиболее абстрактного и общего объекта.

5.4.3. Визуальное программирование

Визуальное программирование существенно облегчает программирование для графического интерфейса типа Windows, который состоит из множества графических объектов: кнопок, окон, меню и т. д. Система ***визуального программирования*** предоставляет программисту:

1) готовую объектную визуальную модель, содержащую множество графических объектов (кнопки, окна, меню, поля ввода и т. д.) и программных модулей, которые их реализуют;

2) среду визуального программирования, которая предоставляет разработчику простой механизм (с использованием манипулятора «мышь») размещения графических объектов на экране.

5.5. Кодирование и исполнение программ

Чтобы создать компьютерную программу, нужно записать алгоритм по специальным правилам на языке программирования (алгоритмическом языке), который понимает как человек, так и компьютер. Такая запись называется исходным кодом программы, или программой. Программа пишется в текстовом редакторе. После чего специальной программой–переводчиком исходный текст на алгоритмическом языке преобразуется в машинные коды, распознаваемые и исполняемые процессором компьютера.

Комплекс программ, включающий программу–переводчика и другие средства написания программ, называется системой программирования.

Компиляция — это процесс перевода текста программы на алгоритмическом языке в машинные коды. Исполнителем процесса компиляции является специальная программа — компилятор.

Возможны два способа компиляции.

– Трансляция. В процессе трансляции происходит преобразование всей программы в машинные коды. Чтобы создать исполняемый файл, необходимо запустить специальный процесс компоновки, который реализуется специальной программой–компоновщиком или редактором связей.

– Благодаря трансляции получается более быстрая по времени работы программа, но выполняемый файл имеет большой объем. Кроме того, выполняемый файл запускается только на компьютере того типа, где программа была транслирована.

– Интерпретация. При интерпретации компьютер «читает» программу по одной строке и сразу выполняет эту строку.

При интерпретации программу не надо переводить всю сразу в машинные коды, и поэтому она имеет маленький объем, равный объему исходного текста программы. Такая программа запускается на любом компьютере, на котором находится соответствующий интерпретатор.

5.6. Ошибки программирования

В процессе разработки программ могут возникать ошибки. Различают два вида ошибок.

1. ***Синтаксические ошибки*** — несоответствие формальным требованиям языка программирования. На них указывает транслятор при трансляции и линковщик при сборке программы.

2. ***Семантические ошибки*** — смысловые ошибки; при них программа «работает», но работает неправильно. Поиск этих ошибок происходит с помощью логического анализа работы программы и ее тестирования.

5.7. Этапы создания программы

Создание любой программы требует разрешения следующих вопросов:

1. Анализ проблемы, на основании которой формулируется постановка задачи.

2. Написание программы.

2.1. Проектирование, или алгоритмизирование — описание требований, разработка алгоритмов.

2.2. Реализация, или кодирование — написание программы, на языке программирования.

3. Устранение ошибок.

3.1. Отладка — устранение синтаксических и других элементарных ошибок в программах на этапах трансляции и сборки.

3.2. Тестирование — проверка правильности работы программы на заранее подготовленных тестах, для которых известен точный результат.

5.8. Описание алгоритма

Описание алгоритма — формализованное представление хода решения задач. Существует несколько форм представления алгоритмов: словесное описание, графическое представление хода решения задачи, запись программы на алгоритмическом языке.

Словесное описание алгоритма — последовательность операций задается перечислением действий алгоритма (алгоритм Евклида, записанный выше).

Графическое представление алгоритма. Наиболее распространенными способами графического представления алгоритмов являются блок–схемы. Блок схема, или диаграмма, кодирует алгоритм наглядными графическими средствами. Она состоит из следующих графических элементов:

1) кружков, кодирующих начало и конец, вход и выход из алгоритма;

2) параллелограммов, описывающих ввод и вывод данных;

3) прямоугольников, в которых описывается действие с данными;

4) ромбов, определяющих проверяемые алгоритмом условия.

Эти фигуры соединяются линиями, указывающими последовательность действий.

На рис. 5.1 приведена блок–схема алгоритма вычисления объема конуса.

Запись алгоритма на языке программирования. Создание и разработка различных языков программирования значительно упростили процесс разработки алгоритмов решения задач, и позволили более широкому кругу специалистов использовать вычислительные машины.

Создание таких языков обусловлено появлением множества задач на стыке областей, для решения которых используются различные методы. Экономические задачи часто приобретают черты научных задач, а научные задачи — черты экономических.

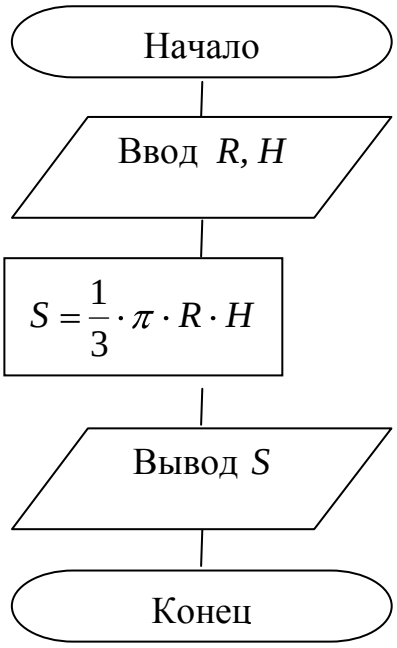


Рис. 5.1. — Блок-схема алгоритма, вычисляющего объем конуса.

Запись программ на языке программирования выполняется с учетом алфавита, синтаксиса и семантики соответствующего языка. Несмотря на обилие языков программирования в записи алгоритма имеется много общего:

- при записи алгоритма следует обозначить начало и конец;
- при описании алгоритма нужно указать величины, которые будут входными при работе этого алгоритма;
- вводимые величины должны быть снабжены описанием, указывающим тип этой величины; на основании этих описаний будет выде-

ляться память для размещения этих величин, определяться совокупность операций, доступных для их обработки;

- необходимо указать величины, которые являются результатом работы алгоритма, и снабдить их описаниями;
- при описании алгоритма необходимо предусмотреть команды, выполняющие определенные действия над введенными величинами.

Таким образом, можно определить общий вид алгоритма:

Название алгоритма
Начало
Команды
Конец

5.9. Элементарные структурные алгоритмы

Следование. Элементарный алгоритм следование — последовательное, линейное выполнение действий. Этот элементарный алгоритм можно изобразить в виде блок–схемы из двух прямоугольников (рис. 5.2).

В виде алгоритма следования может быть изображена любая последовательность операторов, выполняющихся один за другим, имеющая один вход и один выход (рис. 5.3). В языках программирования алгоритм следование соответствует записи операторов ввода и вывода или любого оператора (группы операторов) присваивания.

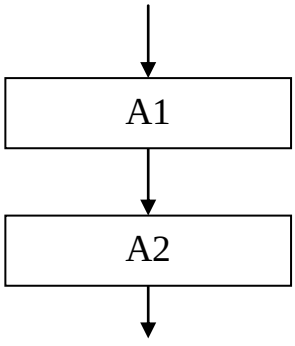


Рис.5.2 — Алгоритм следование

Пример.

Составить алгоритм вычисления площади поверхности *S* и объема *V* цилиндра. Высота цилиндра *H* и радиус основания цилиндра *R* являются заданными величинами.

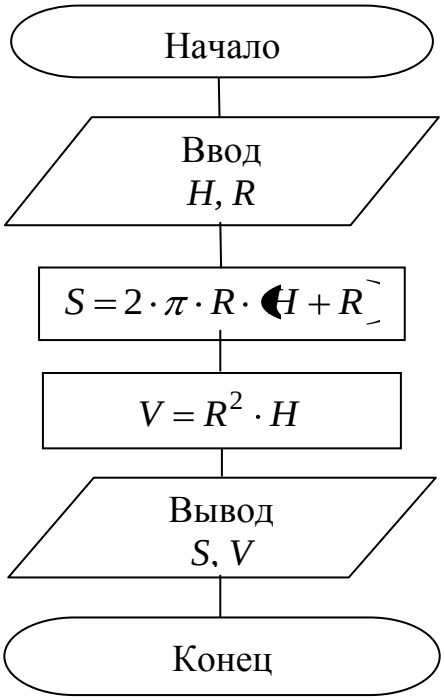


Рис. 5.3 — Алгоритм вычисления площади поверхности и объема цилиндра.

Выбор. Этот алгоритм включает проверку некоторого логического условия (*P*), в зависимости от которого выполняется либо оператор *S1*, либо оператор *S2*

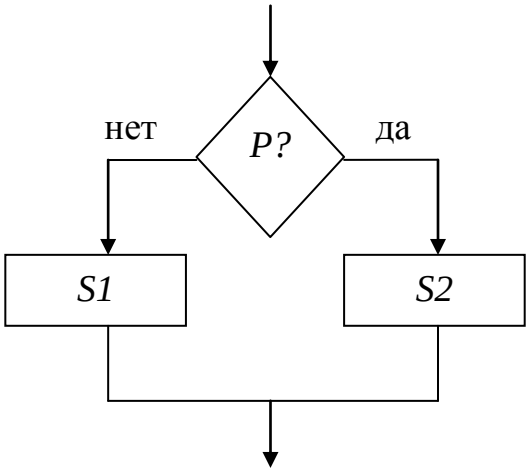


Рис.5.4 — Условная инструкция

Пример.

Составить алгоритм вычисления стоимости *C* покупки с учетом скидки. Скидка в 10% предоставляется, если сумма покупки *S* больше 1000 рублей.

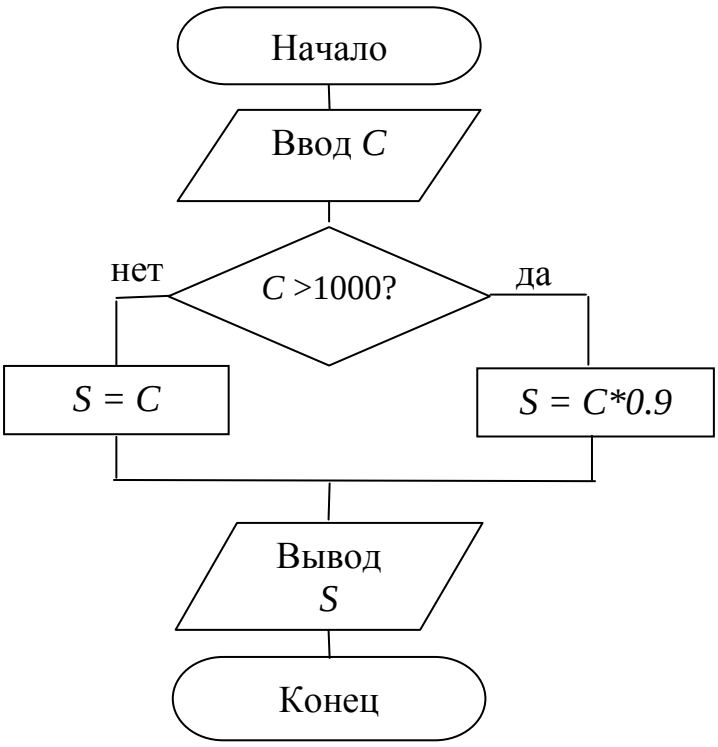


Рис. 5.5. — Вычисление стоимости покупки с учетом скидки

Цикл. Этот алгоритм обеспечивает многократное повторение выполнения оператора *S*, пока выполнено логическое условие *P* (рис. 5.6).

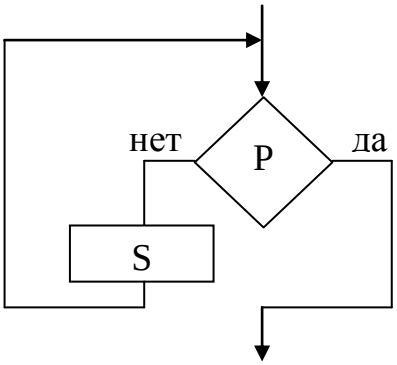


Рис.5.6. Циклическая инструкция

Пример.

Составить алгоритм вычисления суммы S чисел N, вводимых с клавиатуры. Вычисление прекратить, если введено число $N = 0$.

Вычисление суммы K значений является одним из основных приемов программирования. В основе алгоритма вычисления суммы лежит итерационная формула: $S := S + <$*очередной элемент суммирования*$>$. Выбор и суммирование очередного элемента происходит циклически. Для этого требуется выполнить три действия [1]:

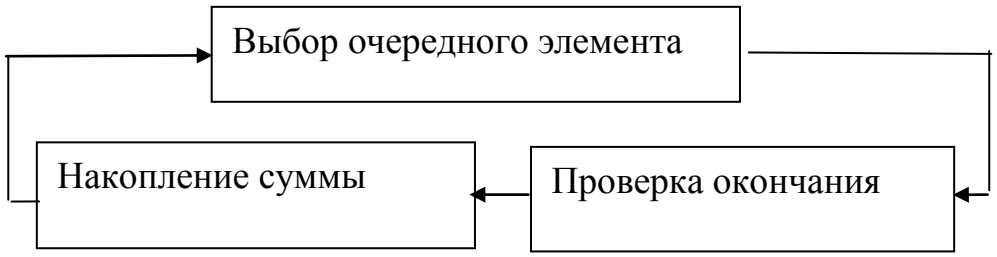


Рис. 5.7. Итерационный процесс накопления суммы элементов.

Правильного результата при выполнении обозначенных действий можно ожидать при условии, что будет определено начальное значение суммы S. В некоторых задачах в качестве начального значения суммы можно установить значение первого элемента суммирования. Но для поставленной задачи $S = 0$ — это то значение, которое гарантирует правильность результата накопления суммы. Алгоритм накопления суммы элементов представлен на рис. 5.8.

Важной особенностью всех перечисленных блоков является то, что каждый из них имеет один вход и один выход.

Кроме того, блок *S*, входящий в состав обобщенного цикла (рис.2.6) и блоки *S1*, *S2*, входящие в состав условной конструкции (рис. 2.4), сами могут быть одним из рассмотренных типов блоков, поэтому возможны конструкции, содержащие «вложенные» блоки. Однако какова бы ни была степень и глубина «вложенности», важно, что любая конст-

рукция в конечном итоге имеет один вход и один выход. Следовательно, любой сложный блок можно рассматривать как «черный ящик» с одним входом и одним выходом.

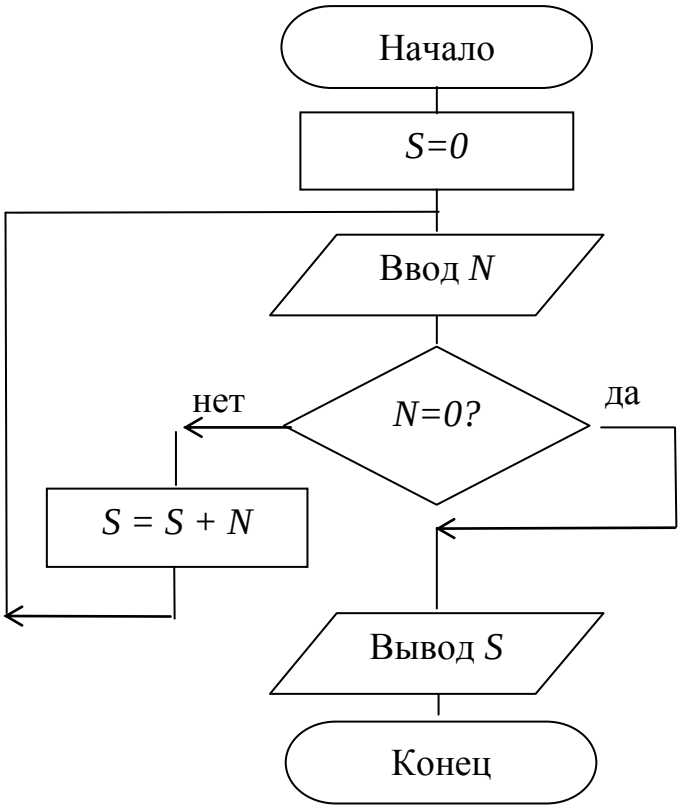


Рис. 5.8 — Алгоритм накопления суммы чисел

При конструировании программы с использованием рассмотренных типов блоков, эти блоки образуют линейную цепочку так, что выход одного блока подсоединяется к входу следующего. Таким образом, программа имеет линейную структуру, причем порядок следования блоков соответствует порядку, в котором они выполняются. Такая структура значительно облегчает чтение и понимание программы, а также упрощает доказательство ее правильности. Так как линейная цепочка блоков может быть сведена к одному блоку, то любая программа может в конечном итоге рассматриваться как единый функциональный блок с одним входом и одним выходом.

Пример решения задачи. Составить программу начисления зарплаты согласно следующему правилу:

Если стаж сотрудника менее 5 лет, то зарплата 2300 руб., при стаже работы от 5 до 15 лет — 5800 руб., при стаже свыше 15 лет зарплата повышается с каждым годом на 500 руб.

Сформулируем задачу в математическом виде:

Вычислить

$$ZP = \begin{cases} 2300, & \text{если } ST < 5; \\ 5800, & \text{если } 5 < ST < 15; \\ 5800 + (ST - 15) \cdot 500, & \text{если } ST > 15. \end{cases}$$

ZP — заработанная плата, *ST* —стаж работы.

Словесное описание алгоритма:

Начало.

1. Ввести значение *ST*.
2. Если *ST* < 5, то *ZP* присвоить значение 2300, перейти в шагу 5.
3. Если *ST* < 15, то *ZP* присвоить значение 5800, перейти к шагу 5.
4. Вычислить $ZP = 5800 + (ST - 15) \cdot 500$.
5. Вывести значение, *ST*.

Конец.

Графическое представление решения поставленной задаче отображено на рис. 5.9.

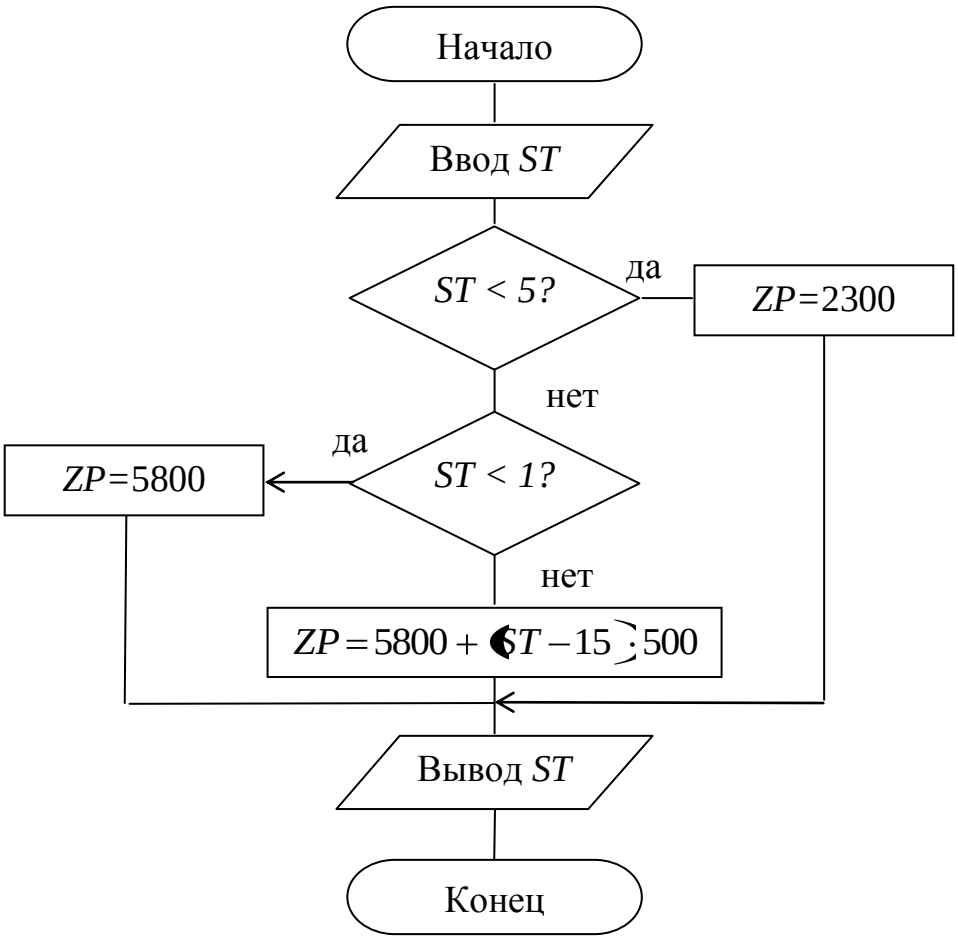


Рис. 5.9 — Алгоритм вычисления заработной платы сотрудника.

5.10. Контрольные вопросы

1. Что такое алгоритм?
2. Каковы способы записи алгоритмов?
3. Кто и когда впервые ввел понятие алгоритма?
4. В чем заключаются основные свойства алгоритма?
5. Перечислите основные алгоритмические структуры и опишите их.
6. Каковы основные принципы разработки алгоритмов?
7. Чем объясняется разнообразие форм записи алгоритмов?
8. Охарактеризуйте словесно-пошаговый способ записи алгоритмов.
9. Что такое результат выполнения алгоритма?

10. Что такое исходные данные?

11. Что представляет собой графическая форма записи алгоритма?

12. Охарактеризуйте основные блоки блок схем?

13. Приведите пример функционального блока.

14. Для чего необходимо ветвление в алгоритмах?

15. Для чего используют циклическую структуру?

16. Что такое тело цикла?

17. Приведите пример функционального блока.

5.11. Задания.

1. Составить алгоритм преобразования введенного дробного числа в денежный формат. Например, число 12.67 должно быть преобразовано к виду 12 руб. 67 коп.

2. Составить алгоритм вычисления корней квадратного уравнения.

3. Заданы действительная и мнимая части комплексного числа $z = x + i \cdot y$. Составить алгоритм преобразования его в тригонометрическую форму $Z = r \cdot (\cos\varphi + i \cdot \sin\varphi)$. $r = \sqrt{x^2 + y^2}$; $\varphi = arctg\,\frac{y}{x}$.

4. Составить алгоритм, в котором выполняется ввод пяти чисел, и после ввода каждого числа выполняет вывод среднего арифметического полученной части последовательности.

5. Составить алгоритм вывода трех чисел в порядке убывания их значений.

6. Заданы три числа: *a*, *b*, *c*. Определить, могут ли они быть сторонами треугольника, и если да, то определить его тип: равносторонний, равнобедренный или разносторонний. Составить алгоритм решения поставленной задачи.

6. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ

6.1. Немного истории

«Языки программирования были одной из первых тем, определивших компьютерную науку, как дисциплину "с собственным лицом". Они не принадлежали к математике, также как и к электронной инженерии. Началом был Алгол 60, своим определением синтаксиса диктовавший жёсткость и точность. То и дело в академии вспыхивали очаги активности, изучались свойства языка, выискивались сбои и противоречия, изобретались мощные алгоритмы синтаксического анализа, преодолевались сложности "общения" с компиляторами. Позже появилось ощущение, что Алгол "узковат" для решения насущных задач. Требовался новый, лучший язык, возможно потомок Алгола. Были созданы специальные комиссии, и разгорелись жаркие споры, кое-кто бредил Грандиозными Формальными Системами, другие думали о вещах, более приближённых к практической среде, к среде, на которой и "взрос" Паскаль.

Паскаль был создан в 1969 г. в духе Алгола 60 с лаконичным синтаксисом, представляющим парадигмы структурированного программирования. Семью годами позже, с появлением микрокомпьютеров, он стал широко известным, и был взят за основу обучения во многих школах и университетах.»[1]

Уже в 1979 году число трансляторов с этого языка перевалило, по оценкам Н.Вирта, за 80. В начале 80-х годов ПАСКАЛЬ ещё более упрочил свои позиции с появлением трансляторов MS-PASCAL и Turbo-PASCAL для персональных ЭВМ. С этого времени язык ПАСКАЛЬ становится одним из наиболее широко используемых языков программирования для персональных ЭВМ.

[1] Никлас Вирт

Язык программирования Паскаль относится к группе универсальных языков программирования, т.е. языков, пригодных для решения любого класса задач (инженерных, экономических, системных) с произвольной структурой обрабатываемой информации.

Язык программирования Паскаль относится к группе языков программирования высокого уровня. Это языки, которые допускают описание задачи в наглядном, легко воспринимаемом виде, языки, которые ориентированы не на систему команд той или иной ЭВМ, а на систему инструкций, характерных для записи алгоритмов того или иного класса.

Язык программирования Паскаль — это язык структурного программирования.

К достоинствам языка программирования Паскаль можно отнести многообразие типов данных и небольшое количество операторов и других конструкций языка, позволяющих, однако, писать сложные программы.

6.2. Алфавит языка

Как и каждый язык программирования, Паскаль имеет свой алфавит. В него входят латинские буквы, цифры от 0 до 9, специальные символы (+, -, *, /, =, ′, ., :, ;. <, >, ^, @, $, #), парные символы (<>, <=, >=, :=, [], {}, (), (* *), (..)), пробелы и зарезервированные слова.

6.3. Структура программы

Программа, написанная на языке программирования Паскаль, состоит из заголовка программы и тела программы (блока), за которым следует точка — признак конца программы. В свою очередь, блок содержит разделы описаний и разделы инструкций.

Структура программы на языке программирования Паскаль в общем виде выглядит следующим образом:

Program *< имя программы>;*

Label *<раздел описания меток>;*

Const *<раздел описания констант>;*

Type *<раздел описания типов>;*

Var *<раздел описания переменных>;*

Procedure *или* ***Function*** *<раздел описания подпрограмм>;*

BEGIN

<раздел инструкций>

END.

На практике при написании программ разделы *const*, *type*, *var*, *label* могут следовать друг за другом в любом порядке и встречаться в разделе описаний сколько угодно раз, а также отсутствовать вообще.

Инструкция *Program < имя программы>* является необязательной.

Раздел инструкций должен присутствовать в любой программе и является основным.

При записи инструкций необходимо соблюдать правила расстановки символов точки с запятой:

- Точка с запятой не ставится в разделах описаний после зарезервированных слов *unit*, *uses*, *label*, *type*, *const*, *var* и ставится после завершения каждого описания.
- Точка с запятой не ставится после слова *begin* и перед словом *end*, так как эти слова являются операторными скобками, а не инструкциями.
- Точка с запятой является разграничителем инструкций, и ее отсутствие между инструкциями вызывает ошибку компиляции.

- В инструкциях цикла точка с запятой не ставится после слов *while, repeat, do* и *until*.
- В условных инструкциях точка с запятой не ставится после слова *then* и перед словом *else*.

6.4. Данные в Паскале

Одним из самых важных аспектов программирования является понятие данных. Данные — это объекты, над которыми выполняются действия. Данные подразделяются на константы и переменные, характеризуемые именем и значением. Информация, хранимая в константе или переменной, называется ее значением. С помощью имени (идентификатора) программист имеет возможность ввести в память компьютера значение соответствующего объекта.

Константы – это объекты с фиксированными не изменяющимися значениями.

Переменные – это объекты с изменяющимися значениями.

При задании имен переменных и констант необходимо соблюдать следующие правила:

- имя должно начинаться с буквы;
- имя не должно содержать пробел, точку, восклицательный знак и символы @, &, $, #;
- длина идентификатора (имени) может быть произвольной, но значащими являются первые 63 символа;
- не рекомендуется задавать имена, совпадающие с ключевыми словами Паскаль;
- имя каждой переменной должно быть уникальным.

6.4.1. Типы данных

Любая константа и любая переменная в языке Паскаль связаны с определенным типом данных. Определение типа данных задает:

- область возможных значений;
- структуру организации данных;
- операции, определенные над данными этого типа.

В Паскале поддерживается определенная классификация типов данных.

Встроенные и определенные пользователем типы данных. Встроенные типы данных изначально принадлежат языку программирования и составляют его базис. Определенные пользователем типы должны быть объявлены в программе. Их описанию предшествует зарезервированное слово *Type*.

```
Type
   <Имя типа> = <type>;
```

Здесь, *<Имя типа>* — идентификатор, определяющий имя типа данных, определенного пользователем, *<type>* – базовый тип.

Например.

```
Type
   Name = String[20];
   F    =  File of Real;
```

Элементарные и сложные типы данных. К элементарным (скалярным) типам данных относятся типы данных, значения которых не содержат составных частей. Это целые типы, логический тип, символьный тип и вещественные типы данных (табл. 6.1), тип–диапазон (отрезочный тип). Сложные или структурированные типы — наборы данных, элемен-

ты которых могут иметь различные характеристики, но по смыслу связаны друг с другом. Это массивы, записи, файлы.

Таблица 6.1

Простые типы данных языка Паскаль

Имя типа	**Возможные значения**	**Требуемая память**
Boolean	True, False	2 байта
Char	Любой символ кодовой таблицы ЭВМ, обрамленный апострофами: 'A', '+', '5' и т.п.	1 байт
Данные, определяющие совокупность целых чисел		
Shortint	–128..128	1 байт
Byte	0..255	1 байт
Integer	–32768..32767	2 байта
Longint	–2147483648..2147483647	4 байта
Word	0..65535	2 байта
Данные, определяющие совокупность вещественных чисел		
Real	2.9e–39..1.7e38	6 байт
Single	1.5e–45..3.4e38	4 байта
Double	5.0e–324..1.7e308	8 байтов
Extended	3.4e–4932..1.1e4932	10 байтов
Comp	–9.2e18..9.2e18	8 байтов

Отрезочный тип данных

Отрезочный тип данных относится к типу, определенному пользователем. Отрезочный тип или тип–диапазон есть подмножество некоторого базового типа. В качестве базового типа не может выступать вещественный тип и тип–диапазон. Описание отрезочного типа имеет следующий формат:

Type

<имя типа> = <нижняя граница> .. <верхняя граница>;

Здесь <*нижняя граница*> — минимальное значение диапазона, <*верхняя граница*> — максимальная граница диапазона.

Примеры отрезочного типа.

```
Type
    N : 1..10;  {ограничение целого типа}
    D : '0'..'9'; {ограничение символьного типа}
```

Присваивание переменным отрезочного типа значений, лежащих вне отрезка, является ошибочным. В остальном отрезочный тип равноправен с типом, значения которого задает отрезок.

6.4.2. Константы

Синтаксис объявления констант:

Const <имя константы> = <константное выражение>;

Тип константы определяется способом записи ее значения. Например:

```
Const
C1 = 50;    { константа типа Integer}
C2 = 3.14; { константа типа Real}
C3 = 'G';   { константа символьного типа – Char}
C5 = False;       { константа логического типа – Boolean}
```

В современных версиях Паскаля допускается объявление типизированной константы. Синтаксис объявления таких констант следующий:

Const <имя константы> [: type] = <константное выражение>;

Пример объявления типизированной константы:

Const F : Boolean = True; {константа логического типа с начальным значением равным True }

Типизированным константам можно присваивать другие значения в ходе выполнения программы, поэтому фактически они представляют собой переменные с начальным значением.

6.4.3. Переменные элементарного типа

Любая переменная в языке Паскаль связана с определенным типом. Эта связь не может изменяться во время исполнения программы. Переменные описываются в разделе описания переменных.

Var

<имя переменной> : <имя типа>;

Каждое объявление переменной связывает имя переменной с ее типом. Если требуется объявить несколько переменных одного типа, то имена этих переменных записываются слева от символа «двоеточие» и разделяются запятыми.

Приведем пример описания переменных необходимых для вычисления стоимости покупки, состоящей из нескольких тетрадей, карандашей и линейки:

```
Var
K, L : Byte; {количество тетрадей и карандашей соответственно}
C1  :  Real; {стоимость одной тетради}
C2  :  Real; {стоимость одного карандаша}
C3  :  Real; {стоимость линейки}
S :  Real; {общая сумма покупки}
```

Переменные *K* и *L* определены как целые положительные числа, поскольку это количественные данные. Переменные *C1, C2, C3* и *S* определены как вещественные числа, так как это стоимость товара, которая может быть задана числом, например, *2.5*, что означает *2* руб. *50* коп.

6.4.4. Контрольные вопросы.

1. Что определяет тип данных?
2. Что такое константа?
3. *Const Alfa = 0.45;* Каким типом определена константа *Alfa*?
4. Описание переменных в Паскале. Привести примеры.
5. Как описать типизированную константу?
6. Как описать тип данных, определенный пользователем?
7. Какие значения может принимать переменная логического типа?
8. Какой тип описывается ключевым словом *CHAR*?
9. Объявите переменные, необходимые для пересчета веса из фунтов в килограммы.
10. Объявите переменные для решения задачи вычисления корней квадратного уравнения.

6.5. Преобразование данных.

Преобразование данных в программе выполняется с использованием набора операций, стандартных процедур и функций и правил их записи.

6.5.1. Операции. Выражения. Правила построения выражений.

В основе программирования на любом алгоритмическом языке лежит построение выражений. Выражение задает правило вычисления зна-

чения переменных. Выражение строится из переменных, констант, встроенных функций с использованием знаков, операций и скобок. Тип выражения совпадает с типом результата.

В таблице 6.2 перечислены основные операции, которые можно использовать в Паскале при построении выражений для обработки числовых данных:

При построении выражений необходимо учитывать следующее:

- арифметические и логические операции выполняются в соответствии с указанным приоритетом;

Таблица 6.2

Операции и их приоритет

Приоритет	**Операция**
1	Унарный минус (–), отрицание (Not)
2	Умножение, деление (*, /), деление нацело (Div), остаток от деления нацело (Mod), конъюнкция (And)
3	Сложение, вычитание (+, –), дизъюнкция (Or).
4	Неравенство (<>), равенство (=), меньше (<), больше (>), меньше или равно (<=), больше или равно (>=)

- одна и та же операция, записанная несколько раз подряд, или операции одного приоритета выполняются слева направо — из двух операций первой выполняется та, которая стоит левее в записи выражения;
- скобки позволяют изменить указанный порядок вычисления выражения, поскольку выражения в скобках имеют наивысший приори-

тет и вычисляются первыми. Внутри скобок действует обычный порядок следования.

Примеры.

1. Записать выражение для вычисления значения переменной $Z = \frac{x-2y}{1+2x}$.

Решение: *Z := (x - 2*y) / (1 + 2*x);*

2. Записать логическое выражение: *X* принадлежит отрезку [2, 5] или [–1, 1].

Решение: *B := (X >= 2) AND (X <= 5) Or (X >= –1) AND (X <= 1)*

6.5.2. Функции обработки числовых данных

Помимо вышеуказанных операций Паскаль предоставляет набор математических функций, расширяющий возможности обработки числовых данных (Табл. 6.3).

Таблица 6.3 Встроенные математические функции

Функция	**Тип аргумента**	**Тип результата**	**Действие**
Abs(x)	целый, вещественный	тип параметра	абсолютное значение аргумента
Arctan(x)	Real	Real	арктангенс (в радианах) аргумента, задающего тангенс угла

Cos(x)	Real	Real	косинус угла. Аргумент *x* задает угол в радианах
Exp(x)	Real	Real	экспонента, т.е. результат возведения *x* в степень числа *e*
Frac(x)	Real	Real	дробная часть числа
Int(x)	Real	Real	целая часть числа
Ln(x)	Real	Real	натуральный логарифм числа
Odd(x)	Integer	Boolean	возвращает True, если x нечетно, и False, если x четно
Pi	–	Real	$\pi = 3.141592653...$
Random	–	Real	равномерно распределенное псевдослучайное число в интервале [0, 1]
Random(x)	Word	Тип параметра	псевдослучайное целое число *0 <= I <= X*.
Randomize	–	–	инициализации датчика псевдослучайных чисел
Round(x)	Real	LongInt	округляет значение вещественного типа до значения целого типа
Sin(x)	Real	Real	синус угла *X*
Sqr(x)	целый, вещественный	Тип параметра	квадрат аргумента
Sqrt(x)	Real	Real	квадратный корень

Пример.

Записать выражение: $H = \left|\cos x - e^{y}\right|^{1+2\ln^2 y}$

Решение.

Предложенная математическая формула содержит пять математических функции (*Abs, Cos, Ln, Exp, Sqr*) и три операции (–, +, возведение в степень). Так как в языке программирования Паскаль нет стандартной функции возведения числа в степень, можно использовать следующую математическую формулу: $x^y = e^{y\cdot\ln(x)}$. По правилам языка это запишется так: *exp(y*ln(x))*. Следовательно решение нашего математического выражения запишется следующим образом:

*H := Exp((1 + 2 * Sqr(Ln(y))) * Ln(Abs(Cos(x) – Exp(y))));*

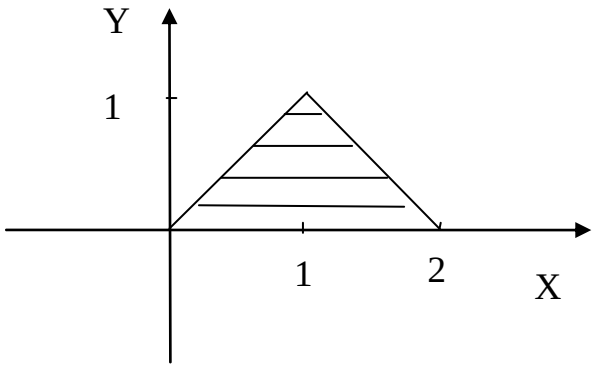
6.5.3. Задания

1. Записать выражение: $H = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \dfrac{x+\dfrac{y}{2}}{2\cdot|x+y|}\cdot Ln(x+1)^{\frac{-1}{\sin(z)}}$

2. Записать выражение: $H = Ln(y^{-\sqrt{x}})\cdot(x-\dfrac{y}{2}) + \sin^2(arctg(z)) + e^{x+y}$

3. Записать выражение: $H = \left|x^{\frac{y}{x}} - \sqrt{\dfrac{y}{x}}\right| + (y-x)\dfrac{\cos y - e^{\frac{z}{y-x}}}{1+(y-x)^2}$

4. Записать логическое выражение, которое определяет принадлежность точки с координатами (x, y) заштрихованной области:



6.6. Операторы

Паскаль — операторный язык. Каждая программа состоит из последовательности операторов. Операторы делятся на две группы: исполняемые и неисполняемые.

Операторы исполняемые служат для описания алгоритма работы программы. Они обеспечивают выполнение логических и расчетных операций; присваивание вычисленных значений некоторым переменным; управление процессом выполнения программ; ввод-вывод исходных и вычисленных значений. Выполнение операторов обработки дает решение запрограммированной задачи. Исполняемые операторы подразделяются на простые и сложные.

Операторы неисполняемые не описывают никаких алгоритмических действий. К ним относится группа декларативных операторов Паскаля, служащих для описания объектов, с которыми работает программа (типов, переменных, констант) и операторы комментария.

При записи текста программ для упрощения чтения, отладки и модификации программы удобней каждый оператор располагать в отдельной строке текста. Следуйте правилу: «Один оператор — одна строка». Но на этой строке разрешается размещать и несколько операторов. Символом разделения двух операторов в одной строке служит символ точка с запятой (;).

6.6.1. Простые операторы

Простые операторы: комментария, присваивания, процедуры, пустой. Они не содержат других операторов.

6.6.1.1. Оператор комментария

Комментарии на исполнение программы не влияют, но необходимы как признак «хорошего стиля». Любые программы используются много-

кратно и не раз модернизируются в процессе жизни. Вы можете сэкономить на комментариях, и написать, а затем и отладить небольшой модуль без них. Но уже через неделю никто, в том числе и Вы, не сможет понять его действие и модифицировать нужным образом. Затраченные на это усилия и время намного превзойдут «экономию». Любая строка текста программы может заканчиваться комментарием. Комментарий в Паскале включает произвольный текст, расположенный внутри фигурных скобок. Обычно в комментариях описываются задачи, решаемые программами; функции, выполняемые процедурами; смысл основных переменных (если он неясен из имен); алгоритмы работы процедур.

Другое применение комментарии находят при отладке программ. Если требуется исключить из программы некоторые операторы (например, вызовы еще нереализованных или сомнительных процедур), то достаточно заключить эти операторы в фигурные скобки, например, при выполнении последовательности операторов:

$$x := x + z;$$
$$\{z := fun(x, z);\}$$
$$y := y * z;$$

Функция *fun* для вычисления нового значения *z* во второй строке не вызывается и вычисление значения переменной y выполняется с учетом старого значения переменной z .

В качестве ограничителей комментария допускается также использование пары символов «(*» и «*)».

Например.

{ Это комментарий}

(Это тоже комментарий*)*

6.6.1.2. Оператор присваивания

Операторы присваивания — основное средство изменения состояния программы (значений переменных и свойств объектов). Они уже

многократно использовались в примерах в предыдущих разделах. Обозначается оператор знаком «:=».

Его синтаксис:

< переменная> := <выражение>;

Переменная определена именем переменной; ***выражение*** задает значение, присваиваемое переменной. Его тип должен соответствовать типу переменной. Нарушение этого условия, например, при попытке присвоить числовой переменной строковое значение, приводит к прерыванию программы.

При выполнении математического оператора присваивания тип выражения соответствует типу переменной в следующих случаях:

- тип выражения совпадает с типом переменной;
- выражение — целого типа, переменная — вещественного типа;
- выражение является переменной, структурно-идентичной переменной в левой части оператора присваивания (массив, запись);

Пример: Известны количество жителей в государстве Албания (3 110 тыс. человек) и площадь его территории (28.7 тыс. кв. км). Определить плотность населения в этом государстве.

Решение.

Поскольку количество жителей — это натуральное число, то для описания этого показателя определим переменную N целого типа. Площадь территории S — вещественное число, т.к. значение этого показателя не обязательно целое число. Для вычисления плотности населения применяем следующую математическую формулу: $P = \frac{N}{S}$. Результат операции деления двух чисел — всегда вещественное число, следовательно, переменная P должна быть объявлена по типу *Real*.

```
Var
S, P : Real;
N    : Integer;
Begin
{определить  начальные значения переменных – оператор присваивания}
S := 28.7;
N := 3110;

{Вычислить плотность населения – оператор присваивания}
P := N / S;

Writeln ('Плотность населения государства Албания',
P:6:1,'чел/кв.км.');
End.
```

Результат работы программы:

Плотность населения государства Албания – 108.4 чел/кв.км.

6.6.1.3. Операторы процедуры

Оператор процедуры имеет следующий формат:

<Имя процедуры > (<список параметров>);

Здесь *список параметров* — перечисленные через запятую имена переменных, констант, выражения. Список параметров может отсутствовать. В этом случае оператор процедуры запишется:

<Имя процедуры >;

Имя процедуры — идентификатор, определяющий имя встроенной процедуры языка программирования Паскаль или имя процедуры, определенной пользователем (см. раздел 3.7).

Рассмотрим процедуры ввода–вывода, обеспечивающие обмен данными между программой и внешним миром. Это процедуры, предназначенные для работы со стандартными текстовыми файлами *INPUT* и *OUTPUT*. Файл *INPUT* связан с клавиатурой, файл *OUTPUT* связан с монитором. Подробнее работа с файлами рассмотрена в разделе 3.8.

Процедура ввода данных

Оператор процедуры ввода данных выполняет инициализацию (присваивание начальных значений) переменных. В языке Паскаль существуют два формата инструкций ввода данных.

Read (<список переменных>);
Readln(<список переменных>);

Список переменных — это перечисленные через запятую имена переменных любого скалярного типа, кроме логического (*Boolean*). Список переменных может отсутствовать.

Инструкция *Read* приостанавливает работу программы и ожидает, когда пользователь введет данные с клавиатуры и нажмет клавишу *Enter*. Разделителями элементов ввода служат символы пробел и нажатие клавиши *Enter*.

Инструкция *Readln* отличается от инструкции *Read* тем, что после ввода значений переменных списка происходит перевод курсора в начало следующей строки, а оставшаяся часть текущей строки данных (если она есть) теряется. То есть следующая инструкция *Read* или *Readln* будет всегда считывать значения с новой строки.

Пример оформления потока вводимых значений.

```
Var
Ch : Char;
X, Y, Z: Integer;
```

```
L   : Real;
Alfa    : String;
Begin
Read(X, Y);
Read (Z); Readln (L);
Readln (Alfa);
Read (Ch);
end.
```

1. С клавиатуры введена последовательность символов, каждая строка заканчивается нажатием клавиши *Enter*:

5 3 –1 3.18 *Enter*
Затерянный мир *Enter*
F *Enter*

Результат: X=5, Y=3, Z=–1, L=3.18, Alfa='Затерянный мир', Ch='f'.

2. С клавиатуры введена последовательность символов:

18 3 87 57.123 Значение строковой переменной * *Enter*
Результат: X=18, Y=3, Z=87, L=57.123, Alfa= ? – ожидается ввод значения.

Если тип данных, вводимых с клавиатуры, не соответствует или не может быть приведен к типу переменных, которые указаны в инструкции *Read (Readln)*, то возникает ошибка ввода-вывода, и программа аварийно завешает работу. При этом на экран выводится сообщение об ошибке: *106 Invalid numeric format* (неверный числовой формат). Использование специальной инструкции — директивы компилятора, которая позволяет отключить автоматический контроль ошибок ввода-вывода, запрещает

прерывание программы, а ошибка описывается встроенной функцией *IoResult*. Данная функция возвращает целое значение, представляющее собой состояние последней выполненной операции ввода-вывода. При этом нулевой код соответствует успешному завершению операции.

Пример использования функции IoResult.

```
Program OSH;
Var
   Alfa : Real;
Begin
{$I-}  {Отключить контроль ошибок ввода-вывода}
   Read (Alfa);
{$I+}  {Включить контроль ошибок ввода-вывода}
   If IoResult = 0  then
   Writeln (Alfa) {Преобразование выполнилось успешно}
   Else
Writeln ('Ошибка преобразования')
End.
```

Оператор процедуры вывода

Процедура вывода предназначена для вывода на экран сообщений и значений переменных. Формат процедуры вывода предусматривает два варианта записи:

```
Write (<список вывода>);
Writeln (<список вывода>);
```

Список вывода может отсутствовать либо содержать перечисленные через запятую:

- имена переменных и констант;

- значения переменных и константы;
- выражения.

Инструкция *Writeln* отличается от инструкции *Write* только тем, что после вывода сообщений или значений переменных курсор переводится на начало следующей строки экрана.

Например.

Writeln (' Результат вычисления =', S);

В данной инструкции список вывода содержит два элемента: строку сообщения, заключенную в апострофы, и переменную S.

*Write (' Площадь круга =', Pi*R*R);*

В данной инструкции список вывода содержит два элемента: строку сообщения, заключенную в апострофы, и математическое выражение, значение которого после вычисление будет отображено на экране.

После любого элемента в списке вывода через двоеточие можно поместить описание поля вывода — формат.

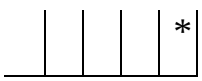
Пример.

В программе написана инструкция:

Write (A : N); {N — целое число }

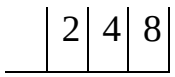
Результат работы даной инструкции, если:

- *A='*'* — переменная типа *Char*, *N=5*.

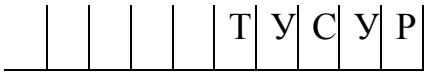


Символ отображается в пятой позиции строки экрана

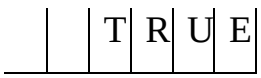
- *A=248* — переменная типа *Integer*, *N=4*.



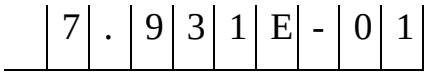
- *A='ТУСУР'* — переменная типа *String, N=10.*



- *A=TRUE* — переменная типа *Boolean, N=6.*



- *A=0.7931* — переменная типа *Real, N=10.*



Для вывода вещественных чисел формат может задаваться двумя целыми числами, разделенными двоеточием.

Write (A : N : M);

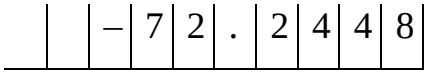
Здесь *N* — ширина поля вывода, *M* — количество символов после десятичной точки.

Например.

……

L := -72.2448; Writeln(L : 10 : 4);

После выполнения данных операторов на экране будет выведено:



6.6.1.4. Контрольные вопросы и задания.

1. Какой формат имеет оператор процедуры ввода данных?
2. Какой формат имеет оператор процедуры ввода данных?

3. Перечислите элементы форматирования вывода информации на экран.

4. Каким переменным нельзя определить начальные значения с помощью процедуры *READ*?

5. Отобразить на экране информацию следующего вида:

Пересчет веса из фунтов в килограммы.

Введите вес в фунтах и нажмите <Enter>.

->

6. В результате вычисления получено значение переменной *L=1.1107142857E+02*. Вывести на экран результат с точностью до второго знака после десятичной точки: *L = 111.07*.

7. На экране отображена последовательность значений данных. Значения введены с клавиатуры через пробел. Определить переменные и оформить оператор ввода данных.

–45 21.78 & 10.4

6.6.2. Сложные операторы

Сложные операторы — составной оператор, условия, варианта, цикла, присоединения. В состав сложного оператора может входить любой оператор языка программирования Паскаль.

6.6.2.1. Составной оператор

Составной оператор задает последовательность операторов, заключенную в операторные скобки (Begin, End).

Операторы, входящие в составной оператор разделяются точкой с запятой. Выполнение составного оператора заключается в последовательном выполнении входящих в него операторов, начиная с первого. Выход из составного оператора осуществляется через закрывающую операторную скобку (End).

Составной оператор играет большую роль при написании других сложных операторов, таких как условный оператор, операторы цикла, присоединения.

Пример составного оператора.

```
Begin
D := Sqrt(B*B–4*A*C);
X1 := (–B + D)/(2*A);
X2 := (B+D)/(2*A);
End;
```

6.6.2.2. Управляющие операторы

Набор управляющих операторов делает честь любому хорошо структурированному языку программирования. Циклы с возможной проверкой условия в начале, в конце и в середине работы оператора, обычный оператор *If* и оператор разбора случаев *Case* — все эти средства позволяют организовать процесс вычислений надежно и эффективно в соответствии с лучшими традициями программирования.

Условный оператор (If Then Else)

Этот общепринятый в языках программирования оператор управления вычислениями позволяет выбирать и выполнять действия в зависимости от истинности некоторого условия.

Имеется два варианта синтаксиса.

```
If <условие> Then <оператор>;
```

```
If <условиие > Then
<оператор1>
Else
<оператор2>;
```

Здесь *условие* обязательно в обоих вариантах. Условие — это выражение, принимающее значение *True* или *False*. Если условие истинно(*True*), то выполняется *оператор1,* если условие ложно(*False*), то выполняется *оператор2*. *Оператор1* и *оператор2* — любые операторы языка Паскаль. Обратите внимание, что после оператора, стоящего перед ключевым словом *Else*, символ точки с запятой не ставится!

Примеры:

1. Найти максимальное и минимальное значение из двух вещественных переменных.

Решение представлено следующим фрагментом программы.

```
Var
  X, Y   : Real; {исходные данные}
Max, Min      : Real; {переменные результата}
.....
If X > Y Then{сравнение значений двух переменных}
 Begin
     Max := X; Min := Y; {если X > Y, то переменной Max присваивается значение переменной X, а переменной Min присваивается значение переменной Y}
 End
Else
 Begin
     Max := Y; Min := X;  {если X < Y, то переменной Max присваивается значение переменной Y, а переменной Min присваивается значение переменной X}
 End;
.....
```

Обратите внимание, что по каждой ветке оператора условия выполняется по два оператора присваивания. Но синтаксис данного опера-

тора позволяет записать только один оператор после ключевого слова *Then* и один оператор после ключевого слова *Else*. Чтобы выполнить данное правило записи условного оператора, операторы присваивания группируются в один составной оператор, записанный межу ключевыми словами *Begin ...End*.

2. Написать программу вычисления площади треугольника, если заданы длины его сторон.

```
Program SQ;
Var
  A, B, C   : Real; {длины сторон}
  P, S      : Real; {полупериметр и площадь}
  R         : Boolean; {переменная условия}
Begin
Write('Введите длины сторон треугольника ');
Readln (A, B, C);
R := (A+B > C) And (A+C > B) And (C+B > A);
If R Then   {если треугольник можно построить}
  Begin
    P := 0.5 *(A+B+C);
    S :=Sqrt(P * (P–A)*(P–B)*(P–C));
    Writeln ('Площадь треугольника равна ', S:6;2);
  End
Else Writeln('Треугольник построить нельзя');
End.
```

В программе введена переменная *R* логического типа (*Boolean*). Данная переменная примет значение равное *TRUE*, если для введенных значений переменных *A, B, C* одновременно выполнятся три условия: *A+B > C* , *A+C > B* и *C+B > A*. Если нарушится хотя бы одно из условий переменная *R* примет значение *FALSE*. Обратите внимание, что в каждой программе

Оператор варианта Case

Этот оператор производит разбор случаев и в зависимости от значения анализируемого выражения выбирает и исполняет один из операторов. Имеет два варианта записи.

Вариант1:

Case <Выражение> of
Список_констант1 : < оператор1>;
Список_констант2 : < оператор2>;
Список_констант3 : < оператор3>;
End;

Вариант 2:

Case <Выражение> of
Список_констант1 : < оператор1>;
Список_констант2 : < оператор2>;
Список_констант3 : < оператор3>;
Else < оператор4>;
End;

Выражение должно присутствовать обязательно. Оно может быть произвольным выражением со значением любого перечислимого типа. *Список_констант* — одно либо несколько значений констант того же типа, что и *выражение.* Список констант можно определять перечислением значений (6, 8, 12) и указанием диапазона значений (2..23).

Пример:

Var
Before : Integer;
CurrentYear : Integer;

```
Str : String;

Begin
{Инициализация переменных:}
CurrentYear := 1999;
Write ('Сколько лет тому назад?'); Readln( Before);
Case CurrentYear – Before of
1954..1969, 1971..1974, 1982 : Str := 'Годы учебы';
1972..1989 : Str := 'Годы воспитания';
Else
Str := 'Прочие годы'
End;
Writeln( Str);
End.
```

Здесь, если *Before = 20*, значением выражения будет 1979, и выполнится работать вариант (*'Годы воспитания '*). При *Before = 25* значение 1974 входит в оба списками, но для исполнения будет выбран лишь первый вариант (*'Годы учебы '*).

6.6.2.2.1. Контрольные вопросы.

1. Какие формы записи имеет оператор условия?
2. В чем отличие условного оператора и оператора варианта?
3. Может ли символ «;» быть прописанным перед ключевым словом *Else*?
4. Что такое условие? Приведите примеры записи условия.
5. Может ли выражение в операторе варианта относится к вещественному типу данных?
6. Как оформит оператор условия, если после ключевого слова *Then* требуется выполнить несколько действий (операторов)?

6.6.2.2.2. Задания.

1. Написать программу вычисления заданной функции

$$Z = \begin{cases} \sqrt{|3y - 5x|}, \text{....если.....} x < y \le 2x \\ \sqrt{|3y + 5x|}, \text{....если.....} y > 2x \end{cases}$$

2. Написать программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: «Рабочий день», «Суббота» или «Воскресенье».

3. Год является високосным, если его номер кратен 4. Из кратных 100 високосными годами являются лишь те года, которые кратны также 400 (например, 1700, 1800, 1900 — не високосные года, 2000 — високосный год). Дано натуральное число N. Определить, является ли високосным год с таким номером.

4. В каждый подарочный набор входит 1 ручка, 2 линейки и 4 тетради. Имеется *a* линеек, *b* тетрадей и *c* ручек. Сколько всего получится подарочных наборов?

5. С клавиатуры вводится две даты. Выдать сообщение «правильно», если первая введенная дата предшествует второй, и «неправильно», если наоборот.

6. Определить, принадлежит ли точка с координатами *X, Y* фигуре, заданной левой полуокружностью вертикальной прямой, проходящей через центр окружности. Координаты центра окружности и радиус произвольны.

7. Написать программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: «Рабочий день», «Суббота», «Воскресенье».

8. Дано натуральное число N ($N < 9999$).

- Является ли это число палиндромом (перевертышем) с учетом четырех цифр, как, например числа 2222, 6116, 0440 и т.д.?

- Верно ли, что число содержит ровно три одинаковые цифры, как, например, числа 6676, 4544, 0006 и т.д.?
- Верно ли, что все четыре цифры числа различны?

9. Дано натуральное число N (N <100), определяющее возраст человека (в годах). Дать для этого числа наименования «год», «года», «лет» и т.д.

10. Даны числа $x_1, x_2, x_3, y_1, y_2, y_3$.Принадлежит ли начало координат треугольнику с вершинами $(x_1, y_2), (x_3, y_1), (x_2, y_3)$?

6.6.2.3. Цикл с параметром (For…)

Операторы цикла служат для организации многократных повторений в программах. В Паскале имеется три вида циклов: цикл с параметром, цикл с предусловием и цикл с условием.

Цикл с параметром позволяет повторять группу операторов заданное число раз. Существует две формы записи оператора цикла с параметром.

1. С увеличением счетчика:

For <счетчик_цикла> := <начало> To <конец> do <оператор>;

2. С уменьшением счетчика:

For <счетчик_цикла> := <начало> DownTo <конец> do <оператор>;

Здесь *счетчик_цикла* — это переменная перечислимого типа. В начале выполнения цикла она принимает значение, задаваемое числовым выражением *начало*. Числовое выражение *конец* задает заключительное значение счетчика цикла. Оно вычисляется до начала исполнения тела цикла и не меняется. Шаг изменения значения переменой *счетчик_цикла* равен 1 для первой формы, шаг равен –1 для второй формы записи цикла. *Оператор (тело цикла)* — любой оператор языка программирования Паскаль.

Если *шаг* цикла положителен, условие завершения цикла:

счетчик_цикла > *конец*

Если *шаг* цикла отрицателен, условие его завершения:

счетчик_цикла < *конец*

Это условие проверяется перед началом работы цикла, а затем — после каждого прибавления *шага* к счетчику цикла. Завершить цикл *For*... можно и с помощью оператора *Break*. Такие операторы могут быть расположены в тех местах тела цикла, где требуется из него выйти, не дожидаясь выполнения условия завершения.

Пример. Вычислить сумму $S = \sum_{i=1}^{20} \frac{x_i}{i}$, где $x_1 = 3,\ x_{i+1} - x_i = 0.5$.

Решение:

```
Var
   X, S : Real;
   I : Integer;
Begin
   X := 3; { начальная установка значения переменной X}
   S := 0; { начальная установка результата}
   For I := 1 to 20 do
     Begin
        S := S + X / I;
        X := X + 0.5;  {выбор очередного элемента}
     End;
   Writeln('Вычисленное значение суммы равно  ',S:8:2);
End.
```

6.6.2.3.1. Задания.

1. Написать программу вычисления факториала заданного числа *n*.

2. Написать программу, которая выводит таблицу квадратов первых десяти целых положительных чисел.

3. Для данного $N > 7$ найти такие целые неотрицательные a и b, $3 \cdot a + 5 \cdot b = N$.

4. Дано действительное число x. Вычислить:

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!}$$

5. Написать программу, которая генерирует три последовательности из десяти случайных чисел в диапазоне от 1 до 10, выводит каждую последовательность на экран и вычисляет среднее арифметическое каждой последовательности. Рекомендуемый вид экрана:

*** Случайные числа ***

6 10 4 2 5 8 1 7 7 3 Сред. арифм. 5.30

10 3 6 1 10 1 3 8 7 6 Сред. арифм. 5.50

5 2 2 5 4 2 2 1 6 10 Сред. арифм. 3.90

6. Написать программу, которая преобразует введенное десятичное число в двоичное.

7. Написать программу, которая преобразует введенное десятичное число в шестнадцатиричное.

8. Написать программу, которая выводит на экран квадрат Пифагора — таблицу умножения.

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	24	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90

9. Треугольником Паскаля называют числовой треугольник, в котором

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      .   .   .   .   .   .   .   .   .
```

по краям стоят единицы, а каждое число внутри равно сумме двух стоящих над ним в ближайшей строке сверху.

Дано натуральное число *n*. Получить первые *n* строк треугольника Паскаля.

10. Натуральное число из n цифр является числом Армстронга, если сумма его цифр, возведенных в n–ю степень равна самому числу (например, $153 = 1^3 + 5^3 + 3^3$) Получить все числа Армстронга, состоящие из двух, трех и четырех чисел.

6.6.2.4. Циклы с условием

Цикл с предусловием (While…)

Этот цикл повторяет выполнение последовательности операторов, пока заданное условие не станет ложным.

Его синтаксис:

While <условие> Do <оператор>;

Здесь *условие* — это логическое выражение, принимающее значение *True* или *False*.

Пример.

Найти наибольшее значение функции $Y = X \cdot e^{B \cdot X - X^2}$ при изменении аргумента от 0 до 4 с шагом 0.1. Коэффициент *B* задается во время работы программы.

Поиск максимума функции будем выполнять по следующему алгоритму. Вначале вычислим первое значение функции при значении аргумента равным 0 и сделаем предположение, что это и есть максимальное число. Запомним его в некоторой переменной *Max*. Далее, вычислим значение функции в следующей точке заданного интервала. Если вновь вычисленное значение окажется больше предыдущего, то запомним его как максимальное, то есть переопределим значение переменной *Max*. И так будем повторять эти действия, пока не переберем все заданные значения аргумента. В результате переменная *Max* будет хранить максимальное значение функции на заданном интервале.

```
Program Maxim;
    Var
    X, B, Y : Real;
    Max : Real;
    H : Real; {Шаг изменения аргумента функции}
Begin
    Writeln(' Введите значение коэффициента B '); Readln(B)
    X := 0;
    H := 0.1;
    Max := X*Exp(B*X–X*X); {максимум – это первое значение}
    While X < 4 do
```

```
    Begin
      X := X + H;     {очередное значение аргумента}
      Y := X*Exp(B*X–X*X); {очередное значение функции}
      If Y > Max Then Max := Y; {переопределить Max }
    End;
  Writeln (' Максимальное значение функции равно ', Max: 8:3);
End.
```

Цикл с постусловием (Repeat …Until …)

Оператор цикла с постусловием используется при описании алгоритмов, в которых заранее неизвестно число повторений. Так как по определению условие определяется после выполнения операторов, составляющих тело цикла, то цикл выполняется хотя бы один раз.

```
Repeat
  тело цикла
Until < условие>;
```

Здесь *условие* является выражением со значениями *True* или *False*. *Тело цикла* — это последовательность операторов, которая будет выполняться, пока *условие* остается ложным.

Пример.

Найти минимальное значение функции $Y = e^{a \cdot x + |c| \cdot x^2}$, при условии, что x изменяется от 0 до 4 с шагом 0.2, значение a=–0.4, значение c= –0.1. Известно, что при заданных условиях функция имеет единственный минимум.

Алгоритм поиска минимального значения функции на заданном интервале подобен алгоритму поиска максимального значения функции, рассмотренному выше. Но в данной задаче возможно сокращение шагов цикла поиска минимума. Ведь как только будет найдено минимальное значение функции, все последующие шаги приведут к вычислению все

большего и большего значения (рис. 3.1). Это ключ к окончанию поиска минимального значения.

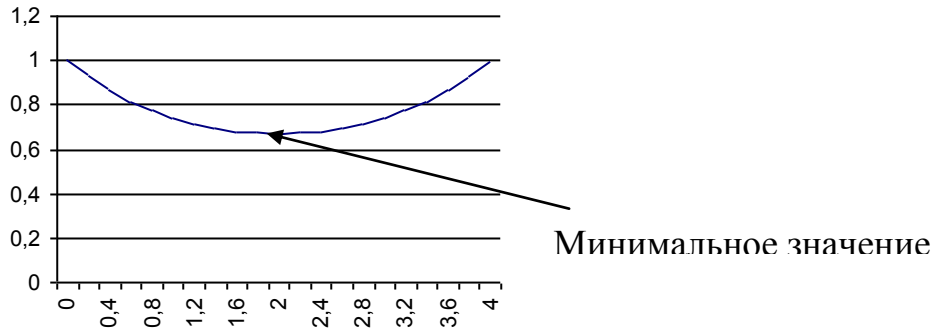


Рис .3.1. Поведение функции на заданном интервале

Идею алгоритма можно описать следующим выражением:

$$Y_{\min} = \begin{cases} Y_i & \text{, если } Y_i < Y_{\min} \\ Прекратить\_поиск & \text{, если } Y_i > Y_{\min} \end{cases},$$

где Y_i — значение функции в очередной точке интервала.

Определим список переменных.

```
Var
   X, A, C  : Real; {исходные данные}
   Y    : Real; {очередное значение функции}
   Min      : Real; {результат}
   H    : Real: {шаг изменения аргумента функции}
   B    : Boolean; {ключ к окончанию поиска}
BEGIN
   X := 0;   H := 0.2;   A := –0.4;    C := –0.1;
   Min := Exp(A * X + Abs(C) * Sqr(X));
   B := False;  {минимум не найден}
   Repeat
     X := X + H;
     Y := Exp(A * X + Abs(C) * Sqr(X));
     If  Y < Min Then Min := Y
```

```
        Else B := True; {найден минимум}
    Until B Or (X > 4);
    Writeln (' Минимальное значение функции равно ', Min:6:2);
End.
```

6.6.2.4.1. Задания.

1. Написать программу, которая определяет максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена). Для завершения ввода определить число 0.

2. Написать программу, которая «задумывает» число в диапазоне от 0 до 10 и предлагает пользователю угадать число за пять попыток.

3. Дано натуральное число.

   - Сколько цифр в заданном числе?
   - Чему равна сумма его цифр?
   - Найти первую цифру числа.

4. В заданном натуральном числе поменять порядок цифр и сравнить полученное число с исходным.

5. Дано натуральное число. Определить, является ли оно степенью пятерки. Например, $625 = 5^4$.

6. Написать программу определения номера члена геометрической прогрессии 2, 4, 8, 16, …, превышающего заданное число z.

7. Имеется фрагмент программы виде инструкции цикла с параметром, обеспечивающий вывод на экран «столбиком» квадратных корней из всех целых чисел от a до b ($a > b$). Оформить этот фрагмент программы в виде:

   - инструкции цикла с предусловием;
   - инструкции цикла с постусловием.

6.6.2.4.2. Контрольные вопросы.

1. Формат цикла с параметром?

2. Формат цикла с предусловием?

3. Формат цикла с постусловием?

4. Какие типы данных допустимы для описания переменных – параметров цикла с параметром?

5. Требуется написать программу, которая вычисляет сумму первых N положительных чисел. Количество суммируемых чисел должно вводится во время работы программы. Какой цикл предпочтителен для решения данной задачи?

6. Требуется написать программу, которая «задумывает» число и предлагает пользователю угадать число за пять попыток. Какой цикл предпочтителен для решения данной задачи?

6.7. Массивы

Язык Паскаль позволяет строить сложные (структурированные) типы данных. Однако самый сложный тип данных, в конечном счете, должен базироваться на небольшом наборе данных элементарного типа. Массив — это самый распространенный структурированный тип данных. Массив представляет собой упорядоченную совокупность данных одного типа, объединенных общим именем. Порядок элементов задается индексами, то есть порядковыми номерами элементов в соответствующей структуре. К необходимости применения массивов мы приходим каждый раз, когда требуется использование целого ряда однотипных величин. Например, в банке получен кредит сроком на N лет, причем для разных лет установлены разные (фиксированные) значения ставок. В этом случае значения ставок для каждого периода (года) удобно рассматривать как совокупность чисел, объединенных в один сложный объект — массив ставок.

Тип массив имеет следующий формат описания:

Type

<имя типа> = ARRAY [<список индексных типов>] OF <имя типа>;

При описании переменных, определенных как массив оформляется следующая инструкция:

Var

<имя переменной> : ARRAY [<список индексных типов>] OF <имя типа>;

В качестве индексных типов можно использовать любые порядковые типы, кроме *LONGINT*. К порядковым типам относятся данные целого, символьного, логического типов. Обычно в списке индексных типов используется тип–диапазон, в котором задаются границы изменения индексов элементов массива. Каждое измерение в списке отделяется запятой и определяется заданием верхней и нижней границы изменения индексов. Наибольшее применение в прикладных задачах нашли одномерные и двумерные массивы.

Например:

```
Var
   A :   Array[0..31] of Char; {одномерный массив A – совокупность 32 элементов символьного типа}
   B :   Array[1..3,1..4] of Real; {двумерный массив B – 12 вещественных чисел. Массив B определен как матрица из трех строк и четырех столбцов}
```

Массивы — это структуры с прямым доступом к элементам. Доступ осуществляется посредством указания имени массива и индекса соответствующего элемента. Индекс элемента прописывается в квадратных скобках, например, *B[3,3]*. Над элементами массива могут выполняться все операции определенные для типа данных, который прописан после ключевого слова *Of*. В целом над массивами можно выполнять только

операцию присваивания *L := C;*, при условии что массивы *L* и *C* относятся к одному типу, определенному инструкцией *Type*.

Пример, демонстрирующий принципы работы с массивами.

Формирование и вывод одномерных массивов.

```
Program Print_Mas;
  Var
Arr1   :     Array[-4..5] of Byte;
Arr2   :     Array[1..10] of Real;
I      :  Integer;
Begin
Writeln (' Элементы массива Arr1 ');
For I := -4 To 5 do
  Begin
Arr1[I] := I + 4;  {заполнение I -го элемента массива}
Write( Arr1[I]:4);{вывод на экран}
end;
Writeln;
Writeln (' Элементы массива Arr2 ');
For I := 1 To 10 do
  Begin
Arr2[I] := I * 0.5;  Write( Arr2[I]:5:1);
end;
End.
```

При исполнении данной программы на экране будет напечатано следующее:

```
Элементы массива  Arr1
  0   1   2   3   4   5   6   7   8   9
 Элементы массива Arr2
  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0
```

6.8. Базовые алгоритмы на массивах.

Обработка массивов — весьма распространенная задача в программировании. Алгоритмы обработки массивов предполагают реализацию того или иного циклического процесса, вычисление индексных выражение, адресацию элементов.

К базовым алгоритмам обработки массивов можно отнести:

- Вычисление суммы элементов числового массива.
- Вычисление произведения элементов числового массива.
- Поиск максимального значения в массиве.
- Поиск минимального значения в массиве.

Все алгоритмы в той или иной задаче рассматривались в данном пособии. Сведем эти сведения в одну систему. Для определенности будем считать, что задан массив $A_1, A_2, A_3, ..., A_n$

6.8.1. Вычисление суммы элементов числового массива.

В основе алгоритма вычисления суммы лежит итерационная формула:

S := S + <очередной элемент суммирования> . Здесь очередной элемент суммирования — элемент массива A_i при изменении значения *i* от 1 до *n*.

С учетом ранее рассмотренной схемы алгоритма, получаем программу:

```
Const n = 50;
Var
    A    : Array[1..n] of integer;
    I    :integer;
    Sum      : integer;
Begin
  Writeln(' Введите элементы массива');
```

```
  For i := 1 to n do Read(A[i]);
  Sum := 0; {обнулить переменную результата}
  {Вычислить сумму n элементов}
  For i := 1 to n do Sum := Sum + A[i];
  Writeln (' Результат суммирования равен ', Sum);
End.
```

6.8.2.Вычисление произведения элементов числового массива.

Акцентируем внимание на том, что алгоритм вычисления произведения элементов должен выполнить те же действия, что и выше рассмотренный алгоритм вычисления суммы. Но итерационная формула, лежащая в основе этого алгоритма преобразуется к виду: *P := P * <очередной элемент суммирования>*. Осталось только определить значение *P* на нулевом шаге, то есть до входа в цикл. Опираясь на тот факт, что, умножив любое число на единицу, мы получим в результате само число, принимаем *P* = 1. Теперь можно записать программу вычисления произведения всех элементов массива.

```
Const n = 50;
Var
    A   : Array[1..n] of integer;
    I   :integer;
    P   : integer;
Begin
  Writeln(' Введите элементы массива');
  For i := 1 to n do Read(A[i]);
  P := 1; {инициализировать переменную результата}
  {Вычислить произведение n элементов}
  For i := 1 to n do P := P * A[i];
  Writeln (' Произведение элементов равно ', P);
End.
```

6.8.3. Поиск минимального и максимального значений в массиве.

Предположим, что требуется найти минимальное значение в массиве чисел. Будем считать, что все числа различны. Как и раньше, мы можем организовать циклический процесс, который позволит последовательно перебрать все элементы массива. Но обработка элемента будет чуть сложнее. Пусть переменная *Min* хранит минимальное из рассмотренных на предыдущих шагах цикла значений элементов массива. Тогда, каждый следующий элемент массива нужно сравнить с минимальным, и если оно оказалось меньше, сделать его значением переменной *Min*. В итоге обработка элемента может быть записана следующим оператором:

```
If A[i] < Min Then Min := A[i];
```

Организуем цикл перебора элементов, записав:

```
For i := 1 to n do
   If A[i] < Min Then Min := A[i];
```

Вновь встает вопрос о выборе значения переменной *Min* на нулевом шаге. Очевидно, что при последовательной обработке всех элементов массива $A_1, A_2, A_3, ..., A_n$ переменная *Min* примет значение равное A_1. В итоге получаем следующую программу:

```
Const n = 50;
Var
    A    : Array[1..n] of integer;
    I    :integer;
    Min      : integer;
Begin
  Writeln(' Введите элементы массива');
  For i := 1 to n do Read(A[i]);
  Min := A[1]; {инициализировать переменную результата}
  {просмотрим оставшиеся элементы, начиная со второго}
  For i := 2 to n do
```

```
    If A[i] < Min Then Min := A[i];
  Writeln (' Минимальное значение массива равно  ', Min);
End.
```

Нетрудно заметить, что в алгоритме поиска максимального значения изменится только условие проверки:

```
If A[i] > Max Then Max := A[i];
```

Часто приходится решать задачу нахождения не только минимального (максимального) значения, но и его позиции в массиве. Следовательно, в алгоритме должна появиться новая переменная *Number* — номер минимального (максимального) элемента. *Number* принимает значение, равное позиции того элемента, который на текущем шаге цикла является минимальным. В итоге алгоритм преобразуется к следующему виду:

```
Const n = 50;
Var
   A      : Array[1..n] of integer;
   I      : Integer;
   Min    : Integer;
   Number: Integer;
Begin
 Writeln(' Введите элементы массива');
 For i := 1 to n do Read(A[i]);
 Min := A[1]; {Минимальный элемент — первый элемент массива}
 Number := 1;{позиция минимального элемента = 1}
 {просмотрим оставшиеся элементы, начиная со второго}
  For i := 2 to n do
    If A[i] < Min Then
     Begin
     Min := A[i];
     Number := i; {найдена новая позиция минимального элемента}
```

```
    End;
  Writeln (' Минимальное значение массива равно  ', Min);
  Writeln (' Позиция минимального элемента равна  ', Number);

End.
```

Следует отметить, что алгоритмы поиска минимального и максимального значений массива применяются не только для числовых, но и для символьных массивов, и массивов строк.

6.8.4. Задания

1. Даны натуральное число N, действительные числа X1, ... , X3n. Вычислить сумму чисел из Xn+1, ..., X3n, которые превосходят по величине все числа X1, ..., Xn.

2. Даны натуральное число N, действительные числа X1, ... , Xn. Выяснить, является ли последовательность X1, ..., Xn упорядоченной

3. Дано натуральное число N и целые числа A1,..., An. Найти номер первого четного члена последовательности A1,..., An; если четных членов нет то ответом должно бить число 0.

4. Даны натуральное число N, целые числа A1,..., A30, B1,..., B40, C1,...Cn. Верно ли, что отрицательный член в последовательности C1,..., Cn встречается раньше, чем в последовательностях A1,...A30 и B1,..., B40? Предполагается, что каждая из последовательностей содержит хотя бы один отрицательный член.

5. Даны A1, A2,... Известно, что A1 > 0 и что среди A2, A3,... есть хотя бы одно отрицательное число. Пусть A1,..., An — члены данной последователь ности, предшествующие первому отрицательному члену (n заранее неизвестно). Получить max(A1^2, ... , An^2).

6. Даны A1, A2,... Известно, что A1 > 0 и что среди A2, A3,... есть хотя бы одно отрицательное число. Пусть A1,..., An — члены дан-

ной последовательности, предшествующие первому отрицательному члену (n заранее неизвестно).

Получить min(A1, 2A2,... , nAn).

7. Даны A1, A2,... Известно, что A1 > 0 и что среди A2, A3,... есть хотя бы одно отрицательное число. Пусть A1,..., An - члены данной последовательности, предшествующие первому отрицательному члену (n заранее неизвестно). Найти количество четных среди A1,...An.

8. У прилавка в магазине выстроилась очередь из n покупателей. Время обслу живания продавцом i-го покупателя равно Ti (i=1,...,n). Пусть даны натуральное n и действительные T1,...,Tn. Получить C1,...,Cn, где Ci - время пребывания i- го покупателя в очереди. Указать номер покупателя, для обслуживания которого продавцу потребовалось самое малое время.

9. Даны натуральное число n, действительные числа A1,..., An. Верно ли, что отрицательных членов последовательности A1,..., An больше, чем положительных ?

10. Даны натуральное число n, действительные числа A1,..., An. Вычислить обратную величину произведения тех членов Ai, данной последовательности, для которых выплнено i+1 < Ai < i!.

11. Дана последовательность x1,x2,...,x15. Вычислить произведение элементов между первым (если их несколько) максимальным и последним минимальным элементами.

6.9. Сортировка массивов.

Под сортировкой информационной структуры (массива, файла) понимают процесс перестановки его элементов для достижения упорядоченности по заданному признаку порядка.

Цель сортировки — облегчить поиск элементов в упорядоченной информационной структуре. В этом смысле элементы сортировки присутствуют почти во всех задачах. Упорядоченные объекты содержатся в

телефонных книгах, в оглавлениях, в библиотеках, в словарях, на складах и так далее (почти всюду, где их приходится разыскивать).

Поэтому методы сортировки очень важны, особенно при обработке данных. При кажущейся простоте задачи именно с сортировкой связаны многие фундаментальные приемы построения алгоритмов, которые и интересуют нас прежде всего.[8]

Методы сортировки обычно разделяют на две группы: сортировка массивов и сортировка последовательных файлов. Идеальным примером огромного разнообразия методов сортировки являются алгоритмы упорядочивания одномерных массивов, которые в свою очередь разбивают на две группы: простые и улучшенные сортировки.

Одномерный массив *A(n)* называют упорядоченным по:

- возрастанию, если для любого $i = 2, 3, 4, \ldots, n \quad a_i > a_{i-1}$
- неубыванию, если для любого $i = 2, 3, 4, \ldots, n \quad a_i \geq a_{i-1}$
- убыванию, если для любого $i = 2, 3, 4, \ldots, n \quad a_i < a_{i-1}$
- невозрастанию, если для любого $i = 2, 3, 4, \ldots, n \quad a_i \leq a_{i-1}$

Практически в каждом алгоритме сортировки массива можно выделить три ключевых момента:

1. сравнение, определяющее упорядоченность пары элементов;
2. перестановку, меняющую местами пару элементов;
3. собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов до тех пор, пока все элементы множества не будут упорядочены.

Выбор алгоритма сортировки для конкретной задачи выполняется с учетом эффективности соответствующего алгоритма. Удобной оценкой эффективности является количество *C* необходимых сравнений и является функцией от числа элементов *n*. Улучшенные алгоритмы сортировки требуют порядка $n \cdot \ln_2 n$ сравнений. Для простых методов сортировки $C \sim n^2$.

6.9.1. Простые методы сортировки

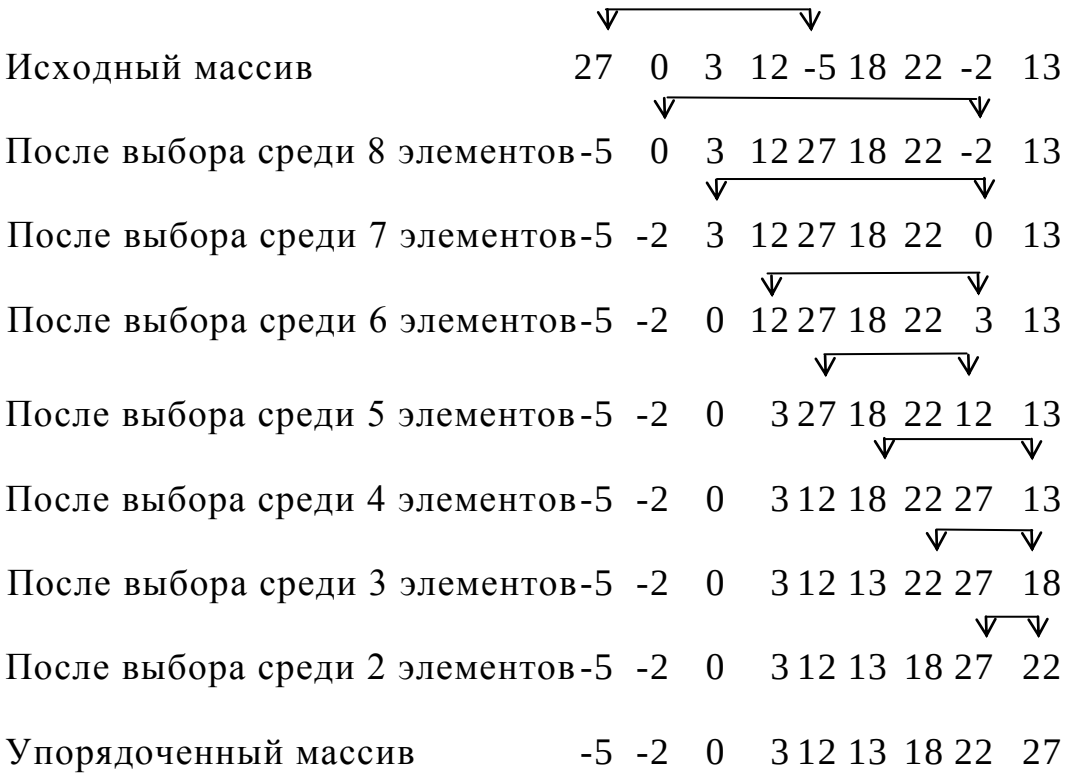
Простые методы сортировки эффективны для массивов с достаточно малым значением n. К простым методам сортировки относятся: сортировка выбором, обменная сортировка, сортировка простыми вставками.

Сортировка выбором.

Для определенности будем упорядочивать по возрастанию массив $A_1, A_2, A_3, ..., A_n$.

Идея метода такова. Найдем минимальный элемент массива и поставим его на первое место. Элемент, стоявший на первом месте пока перенесем на место минимального элемента. Далее найдем минимальное значение среди оставшихся $n-1$ элементов и поставим его на второе место. И так будем выполнять аналогичные действия для $n-2, n-3, ..., n-n+1$ элементов. Последний оставшийся элемент — наибольший элемент данного массива.

Поясним идею метода на примере.



Вспомним алгоритм поиска минимального элемента массива. Это позволит сделать первый набросок программы. Введем переменную *i*, в которую запомним номер первого элемента в неупорядоченной последовательности. Она определяет и номер элемента сравнения.

```
k :=i; {минимальный элемент – A[ k]}
For j := i+1  to n do
If A[k] > A[j] then k := j; { минимальный элемент на j – ой позиции}
```

Отметим, что номер первого элемента *i* неупорядоченной последовательности равен номеру очередного этапа поиска минимального элемента. Кроме того, поиск минимального элемента выполнялся $n-1$ раз. Следовательно, можно записать:

```
For i := 1 to n-1 do
  Begin
     k := i;
.....
```

Получаем следующий набросок программы:

```
For i := 1 to n-1 do
  Begin
    k := i;
    For j := i+1 to n do
      If A[k] > A[j] then k := j;
  End;
```

Не выполнена только перестановка значений, которую можно записать так:

```
B := A[k];  {запомнить минимальный элемент в буферной переменной}
```

A[k] := A[i]; {на место минимального элемента записать первый элемент неупорядоченной последовательности}

A[i] := B; {поставить минимальный элемент в начало неупорядоченной последовательности}

А теперь запишем алгоритм полностью:

```
For i := 1 to n-1 do
  Begin
    k := i;
    For j := i+1 to n do
      If A[k] > A[j] then k := j;
    B := A[k];
    A[k] := A[i];
    A[i] := B;
  end;
```

Обменная сортировка («пузырька»).

Идея этого метода отражена в его названии. Самые легкие элементы массива «всплывают» наверх, самые «тяжелые» — тонут. Идею метода можно сформулировать так: просматриваем весь массив «снизу вверх» и меняем стоящие рядом элементы в том случае, если «нижний» элемент меньше, чем «верхний». Таким образом, мы вытолкнем наверх самый «легкий» элемент всего массива. Теперь повторим всю операцию для оставшихся неотсортированными N-1 элементов (т.е. для тех, которые лежат «ниже» первого. И так далее, пока не останется один элемент. Как видно, алгоритм достаточно прост.

Вновь поясним на примере (рис. 3.2) .

Алгоритм выполняет $n-1$ раз просмотр элементов неупорядоченной последовательности. Каждый раз осуществляется перебор всех элемен-

тов, начиная с последнего и заканчивая первым элементом неупорядоченной последовательности.

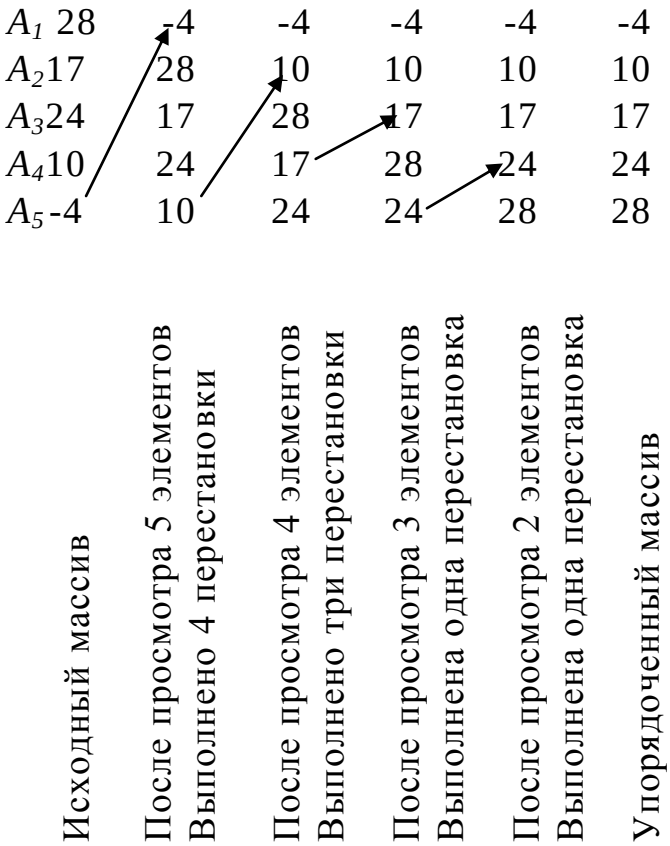


Рис. 3.2. Обменная сортировка

Эта последовательность операций может быть записана следующим программным кодом:

```
For i := 1 to n-1 do  {1, 2, 3, ... n-1  просмотр массива}
    begin
    For j := n-1 downto i do
        If A[j] > A[j+1] then {сравнение рядом стоящих элемен-
тов}
       Begin
         B := A[j];
         A[j] := A[j+1];
         A[j+1] := B;
       end;
```

Алгоритм вставками

Идея этого алгоритма основана на предположении о том, что в исходном массиве $A_1, A_2, A_3, ..., A_n$ первые $i-1$ элементов уже упорядочены. Тогда, выбрав i элемент, выполняем просмотр ранее упорядоченной последовательности, пока не найдется некоторый элемент $A_k > A_i$. Затем сдвинем вправо на одну позицию все элементы последовательности $A_{k+1}....A_{i-1}$ и на освободившееся место поставим A_i.

Полагаем $i = 2$. В этом случае упорядоченная последовательность состоит из одного элемента. Тогда можно сделать первый набросок программы:

```
For i := 2 to n do
 Begin
   B := A[i]; { запоминаем элемент вставки}
For k := 1 to i-1 do
     If A[i] <= A[k] then
{найдено место вставки = k;
 выполнить сдвиг элементов A[k+1]....A[i-1] вправо;
вставить A[i] элемент на место A[k] }
End;
```

Для выполнения сдвига элементов массива запишем следующий оператор цикла:

```
For j := i downto k+1 do  A[j] := A[j-1];
```

После выполнения данного оператора значение элемента A_k будет равно значению элемента A_{k+1}. Следовательно, можно выполнить оператор *A[k] := B*;, то есть на *k* позицию записать элемент вставки

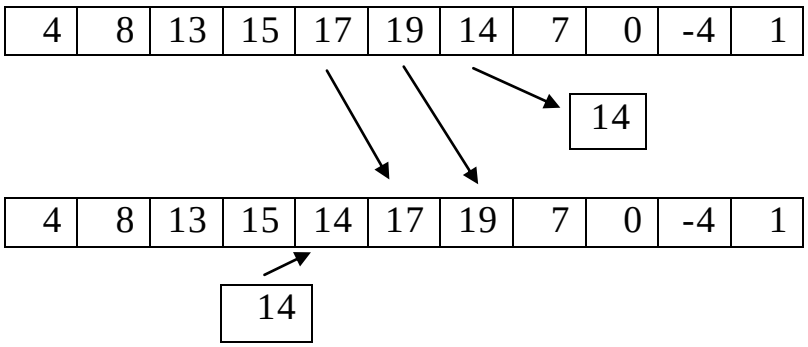


Рис. 3.3. Сдвиг элементов массива при значении $i = 7$, $k = 5$.

Приведем полный текст алгоритма вставками.

```
For i := 2 to n do
  Begin
   B := A[i];

  For k := 1 to i-1 do
    If A[i] <= A[k] then
    Begin
      For j := i downto k+1 do A[j] := A[j-1];
      A[k] := B;
    End;
  End;
```

6.9.2. Усовершенствованные сортировки

Рассмотрим два метода сортировок: сортировку Шелла и сортировку бинарными вставками, в основе которых лежат принципы ранее рассмотренных сортировок.

Сортировка Шелла

Сортировка Шелла является усовершенствованием обменной сортировки. Основная идея этого алгоритма заключается в том, что вначале устраняется массовый беспорядок в массиве путем сравнения и перестановки далеко стоящих друг от друга элементов. Расстояние между элементами задается значением шага H, который изменяется после каждого прохода по массиву. Величина шага изменяется по правилу: $H_{i+1} = \frac{H_{i-1}}{2}$ для массивов, содержащих более 500 элементов и $H_{i+1} = \frac{H_{i-1}}{3}$ для массивов, содержащих менее 500 элементов. За H_0 принимается число элементов в массиве.

Как видно, интервал между сравниваемыми элементами постепенно уменьшается до единицы. Это означает, что на поздних стадиях сортировка сводится просто к перестановкам соседних элементов (если, конечно, такие перестановки являются необходимыми).

Приведем пример программы, в которой реализован алгоритм сортировки Шелла. Алгоритм выполнен в процедуре *Shell*.

```
Type
  Massiv =  Array[1..100] Of Integer;
var
   A     : Massiv;
   i, n  : integer;
Procedure Shell ( Var A : Massiv; n : integer);
Var
   i, j, h, k  : integer;
   B           : integer;
Begin
 If n > 500 then h := (n-1) div 2 {вычисляем первое значение шага}
   else h := (n-1) div 3;
```

```
  While h > 0  do
  begin
   for i := 1 to h do
    Begin
     for k :=1 to n div h do  {модификация алгоритма «пузырька»}
       begin          {h = 1 — обменная сортировка}
        j := i+ k - 1;
        While (j+h) <= n do
          Begin
            If A[j] > A[j+h] then
              Begin
                b := A[j];
                A[j] := A[j+h];
                A[j+h] := B;
              End;
            j := j +h;
          end;
       end;
     end;
     If h = 1 then Exit; { если шаг = 1, то завершить алгоритм}
       If n > 500 then h := (h-1) div 2 + 1
       else h := (h-1) div 3+1;
   end;
End;

Begin
 Writeln('Введите количество элементов массива'); Readln(n);
Randomize;
  Writeln('Элементы неупорядоченного массива');
  For i := 1 to n do
```

```
    Begin
      A[i] := Random(35)-6;
      Write(A[i]:4);
    end;
  Writeln;        Writeln;
  Shell(A, n); вызов процедуры сортировки
  Writeln('Элементы упорядоченного массива');
  For i := 1 to n do Write(A[i]:4);
  Writeln;
  Readln;
end.
```

Сортировка бинарными вставками.

Алгоритм сортировки бинарными вставками является усовершенствованием сортировки простыми вставками. Аналогично сортировке вставками полагаем, что в исходном массиве $A_1, A_2, A_3, ..., A_n$ $i-1$ элементов упорядочены. Тогда выбираем элемент вставки A_i. Далее, чтобы найти место вставки в упорядоченной последовательности ищем серединное значение A_s. Если серединное значение равно A_i, то место вставки для A_i. Иначе поиск места вставки продолжается по следующей схеме: если $A_s < A_i$, то изменяем правую границу упорядоченной последовательности (правая граница = *s)*, если $A_s \geq A_i$, изменяем левую границу (правая граница = $s+1$). Таким образом, проходит сужение интервала поиска места вставки. Сужения интервала продолжаем, пока его длина не станет равной 1 — это и есть место вставки.

Поясним на примере.

A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
-3	3	5	7	8	9	21	22	13	27

Задача: найти место вставки элемента A_9.

1. Определи границы интервала поиска. Правая граница p = 8, левая граница q = 1.

2. Вычислим середину интервала s = 4 (s — результат целочисленного деления $\frac{p+q}{2}$)

3. Так как $A_9 > A_4 (13 > 8)$, переопределяем левую границу q=5. Интервал сужен до 4 элементов.

4. Вновь вычисляем середину интервала s = 6.

5. Так как $A_9 > A_6 (13 > 9)$, переопределяем левую границу q=7. Интервал сужен до 2 элементов.

6. Вновь вычисляем середину интервала s = 7.

7. Так как $A_9 < A_7 (13 < 21)$, переопределяем правую границу p = 7. Интервал сужен до 1 элемента, следовательно, место вставки найдено.

А теперь запишем алгоритм бинарной вставки в виде процедуры на алгоритмическом язык Паскаль.

```
Procedure Binar ( Var A : Massiv; n : integer);
Var
  i, j, p, k, q, s  : integer;
  B                 : integer;
Begin
  For i := 2 to n do {поиск места для i-го элемента}
   begin
    For j := 1 to i-1 do { в упорядоченной последовательности}
      Begin
       p := 1; {правая граница интервала}
       q := i-1;{левая граница интервала}
       While p < q do
        begin
          s := (p+q) div 2;  {вычисление середины интервала}
```

```
        If A[s] = A[i] then
          Begin
           p := s; Break; {выход из цикла поиска места вставки}
          End
        Else
          If A[s] > A[i] Then q := s else p := s+1;
      end;
      If A[p] >= A[i] then
        Begin
          B := A[i]; {запомнить элемент вставки}
      {сдвиг элементов от p+1 до i}
          For k := i downto p+1 do A[k] := A[k-1];
          A[p] := B;
        end;
    End;
  End;
```

6.9.3. Задания

1. Дана последовательность a1, a2, ..., a40. Расположить положительные элементы последовательности, стоящие на нечетных местах, по возрастанию.

2. Дана последовательность a1, a2, ..., a100. Расположить ненулевые элементы, стоящие на нечетных местах по убыванию.

3. Дана последовательность x1, x2, ..., x100. Элементы, стоящие на нечетных местах, расположить в порядке возрастания, а на четных - в порядке убывания.

4. Дана последовательность R1, R2,..., R50, элементы которой есть целые двузначные числа. Упорядочить последовательность по возрастанию сумм цифр соответствующих элементов.

5. Дана последовательность a1, a2,..., a50. Требуется упорядочить ее по возрастанию абсолютных значений элементов, кратных трем

6. Дана последовательность x1, x2,..., x50. Требуется расположить отрицательные элементы последовательности в порядке убывания.

7. Дана последовательность x1, x2, ..., x40. Требуется упорядочить ее по убыванию произведение цифр соответствующих элементов, если эти элементы последовательности есть целые двузначные числа.

8. Дана последовательность x1, x2, ..., x40. Требуется расположить элементы, попадающие в интервал [a,b] в порядке возрастания.

6.10. Задача поиска.

Задача поиска состоит в отыскании в последовательности элемента (или нескольких элементов), равного некоторому заданному значению. Поиск, как и сортировка, находит применение практически во всех прикладных программах. Процесс поиска во многом зависит от организации массива данных. Поиск последовательным перебором осуществляется в неупорядоченных структурах. Если же данные упорядочены, то применяют более эффективные методы, например, метод бинарного поиска.

Поиск последовательным перебором.

Алгоритм поиска последовательным перебором всех значений во многом схож с алгоритмом поиска минимального или максимального элемента массива. Отличие заключается в том, что в качестве эталона сравнения будет выступать некоторая переменная L, назовем ее ключем поиска. Вновь считаем, что задан некоторый массив $A_1, A_2, A_3, ..., A_n$. Приведем пример записи алгоритма поиска позиции элемента в неупорядоченном массиве, значение которого равно L..

```
Const n = 50;
Var
```

```
A       : Array[1..n] of integer;
I, L    :integer;
k       : integer;
Begin
For i := 1 to n do A[i] := Random (40) - 9;
Write(' Введите  искомое значение ');
Readln(L);
k := 0; { элемент не найден}
For i := 1 to n do
 if A[i] = L Then k := i;
If k <> 0 Then
  Writeln (' Позиция искомого элемента равна ',k)
Else Writeln (' Элемент с заданным значением не найден');
End.
```

Бинарный поиск.

Бинарный поиск или поиск методом деления пополам является одним из эффективных методов поиска в больших отсортированных массивах. Основная идея алгоритма бинарного поиска реализует идею, заложенную в алгоритм сортировки бинарными вставками. Учитывается условие упорядоченности массива. Тогда вместо последовательного просмотра всех элементов массива находится срединный элемент. Сравнивается его с ключом поиска. Результат сравнения позволяет сделать вывод: в какой половине может находиться искомое значение. Далее переопределяется интервал дальнейшего поиска. Далее процесс повторятся, и так до тех пор, пока интервал поиска не сузится до 1 элемента.

Пусть задан массив $A_1, A_2, A_3, ..., A_n$, упорядоченный по возрастанию. Чтобы найти позицию искомого элемента напишем следующую процедуру.

```
Procedure Poisk ( Var A : Massiv; n : Integer;
                                  L : Integer; Var k : integer);
Var
  i,  q, p, s      : Integer;
  B                : Integer;
  Flag             : Boolean;

Begin
        p := 1; q := n;
       Flag := False;  k := 0; { Элемент не найден}
        While (p < q) and Not Flag do
        begin
          s := (p+q) div 2;
          If A[s] = L then
            Flag := True
          Else
            If A[s] > L Then q := s-1 {сужаем интервал поиска}

            Else p := s+1;
         end;
         If Flag Then  {если найдено искомое значение}
           If p=q then k := p Else k := s;
end;
```

6.10.1. Вопросы и задания.

1. В чем заключается задача поиска?
2. Сформулируйте идею бинарного поиска?
3. Какой поиск можно выполнить на неупорядоченном массиве?
4. Напишите программу поиска в массиве строк.

5. Как изменить программу поиска бинарными вставками, если исходный массив упорядочен по убыванию?

6.11. Строки.

Строка в Паскале трактуется как цепочка символов. Максимально допустимое количество символов равно 255. Объявление переменных строкового типа имеет следующий формат:

```
Var
  Stroka : String; {строка длиной до 255 символов}
  St     : String[20];{ строка длиной до 20 символов}
```

Строка трактуется как массив символов переменной длины. К любому элементу строки можно обратиться как к элементу одномерного массива. Например, *St[5] := 'Q';*.

Значение констант и переменных строкового типа в тексте программы заключаются в апострофы. Ввод строковых переменных с клавиатуры осуществляется оператором ввода ***Readln***.

Над строковыми переменными, определенными в программах на алгоритмическом языке Паскаль допустимо выполнение двух видов операций: сравнения и конкатенации строк.

Операция конкатенации используется для сцепления двух или нескольких строк. Обозначается данная операция знаком «+».

Например:

```
Var Stroka As String
…….
Stroka = ' Формируется  ' + 'новая  ' + ' строка ' ;
Writeln(Stroka)
```

При сравнении строк применимы обычные операции сравнения. При этом сравнение может быть осуществлено в соответствии с расположением строк в словаре.

Основные функции обработки строковых переменных

CONCAT(S1 [,S2, ... ,SN]) — функция типа *STRING;* возвращает строку, представляющую собой сцепление строк-параметров *SI, S2,..., SN*.

COPY(ST, INDEX, COUNT) — функция типа STRING; копирует из строки *ST COUNT* символов, начиная с символа с номером *INDEX*.

DELETE(ST, INDEX, COUNT) — процедура; удаляет *COUNT* символов из строки *ST*, начиная с символа с номером *INDEX*.

INSERT(SUBST, ST, INDEX) — процедура; вставляет подстроку SUBST в строку *ST*, начиная с символа с номером *INDEX*.

LENGTH(ST) — функция типа *INTEGER*; возвращает длину строки *ST*.

POS(SUBST, ST) — функция типа *INTEGER*; отыскивает в строке ST первое вхождение подстроки *SUBST* и возвращает номер позиции, с которой она начинается; если подстрока не найдена, возвращается ноль.

STR(X [,WIDTH [:DECIMALS]], ST) — процедура; преобразует число — любого вещественного или целого типов в строку символов *ST* так, как это делает процедура *WRITELN* перед выводом; параметры *WIDTH* и *DECIMALS*, если они присутствуют, задают формат преобразования: *WIDTH* определяет общую ширину поля, выделенного под соответствующее символьное представление вещественного или целого числа *X*, а *DECIMALS* — количество символов в дробной части (имеет смысл только в том случае, когда *X* - вещественное число).

VAL(ST, X, CODE) — процедура; преобразует строку символов *ST* во внутреннее представление целой или вещественной переменной *X*, которое определяется типом этой переменной; параметр *CODE* содержит ноль, если преобразование прошло успешно, и тогда в *X* помещается результат преобразования, в противном случае он содержит номер позиции в строке *ST*, где обнаружен ошибочный символ, и в этом случае содержимое *X* не меняется; ведущие пробелы в строке *ST* должны отсутствовать.

UPCASE(CH) — функция типа *CHAR*; возвращает для символьного выражения *CH*, которое должно представлять собой строчную латинскую букву, соответствующую заглавную букву; если значением *CH* является любой другой символ, функция возвращает его без преобразований.

Пример.

Дана строка символов. Исключить из нее группы символов, расположенные между круглыми скобками. Сами скобки тоже исключить.

Решение.

```
Var
 Left, Right: Byte; {позиция левой скобки и правой скобки}
St        : String; {исходная строка}
Begin
Write (' Ведите строку символов '); Readln(St);
While (Pos('(', St) <> 0) And (Pos(')', St) <> 0) do
Begin
Left := Pos('(', St); { найти позицию первой левой скобки}
Right := Pos(')', St); { найти позицию первой правой скобки }
 Delete := (St, Left, Right-Left-1); {удалить все символы между
                                      скобками, включая скобки}
End;
```

```
Writeln('', St);
End.
```

Цикл удаления элементов строки завершит работу, когда в строке не будет найдена пара круглых скобок. Это приведет к тому, что любая из функций *Pos('(', St) Pos* или *Pos(')', St)* вернет значение, равное нулю.

6.11.1. Задания

1. Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Найти длину самого короткого и самого длинного слова.
2. Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Определить, сколько в строке слов, заканчивающихся буквой а.
3. Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Найти количество слов в строке, у которых первый и последний символ совпадают.
4. Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Удалить из каждого слова строки все последующие вхождения его первой буквы.
5. Задана строка символов. Подсчитать, сколько различных символов встречается в ней. Вывести эти символы на экран.
6. Задана строка символов. Группы символов, разделенные пробелом и не содержащие пробелов внутри себя, будем называть словами. Определить количество слов, которые содержат ровно две буквы а.
7. Написать процедуру, которая преобразует текст, оставляя только один пробел между словами.

8. Дана строка текста, между словами текста минимум один пробел. Написать программу, которая между словами текста ставит по три знака точка (…) и подсчитывает количество замен.

9. Написать программу, которая преобразует строку текста, выравнивая ее до заданной длины добавлением пробелов между словами.

10. Написать программу, которая перевернет введенную с клавиатуры фразу.

6.12. Записи.

Запись относится к сложным типам данных. Запись — структура данных, состоящая из фиксированного числа элементов, называемых полями записи. В отличие от массива элементы записи могут быть различного типа. Запись хорошо ассоциируется с таблицей. В этом случае имена полей записи соответствуют наименованиям столбцов таблицы, а значения элементов (полей) записи — значениям соответствующих ячеек выделенной строки таблицы.

Синтаксис объявления типа запись:

```
Type
    <имя типа> = Record
<имя поля> : <type>;
<имя поля> : <type>;
End;
```

Переменная типа запись объявляется в разделе описания переменных:

```
Var
    <имя переменной> : Record
<имя поля> : <type>;
<имя поля> : <type>;
End;
```

Здесь <*type*> — имя любого ранее определенного типа данных, включая тип запись. Обрабатываются записи поэлементно. Для доступа к элементу записи используется составное имя, которое состоит из имени записи и имени соответствующего поля или полей. Имена разделяются знаком точка. Над элементами записи (полями) можно выполнять все операции, которые допустимы над данными соответствующего типа. В целом над записью можно выполнять только операцию присваивания.

Пример.

Определить сумму отчислений по каждому оптовому покупателю. Отчисления по сумме каждого платежа определяются следующим образом: до 500 рублей – 1.5%, свыше 500 рублей – 3%. Оптовый покупатель имеет следующие реквизиты: код покупателя, фамилия, сумма платежа.

Решение:

Информация, связанная с данной задачей может быть представлена в следующем виде:

Таблица 6.4.

Ведомость расчетов с оптовыми покупателями.

Код покупателя	Фамилия	Сумма платежа	Отчисления
700045	Иванов	458	6.87
700321	Шведова	1298	38.94
…	…	…	…

Каждая строка представленной таблицы — есть объединение разнотипных данных. В этом случае, чтобы сохранить целостность воспри-

ятия данных, связанных с одним покупателем, целесообразно использовать специфическую структуру данных — запись, а для описания всей таблицы — массив записей.

```
{Определение структуры записи из четырех полей}
    Type
      Person = Record
             Kod  : Longint; { поле – код покупателя}
             Fam  : String[18]; { поле – фамилия покупателя}
             Sum  : Real; { поле – сумма платежа}
             Tax  : Real; { поле – отчисления }
            end;
{Объявление массива записей(покупателей)}
    Var
      Group : Array[1..10] Of Person; {массив покупателей}
      i     : Integer;
    Begin
      For i := 1 to 10 do {для каждого покупателя}
{Присваиваются значения полям соответствующих записей}
        Begin
          Write('Введите код покупателя '); Readln(Group[i].Kod);
          Write('Введите фамилию покупателя ');
          Readln(Group[i].Fam);

          Write('Введите сумму платежа ');
          Readln(Group[i].Sum);
          If Group[i].sum >= 500 then
               Group[i].Tax := Group[i].sum * 0.015
        Else Group[i].Tax := Group[i].Sum * 0.03;
      End;
```

```
{А теперь просмотр результата на экране монитора}
 Writeln ('Ведомость расчета с оптовыми покупателями');
Writeln ('  Код          Фамилия    Сумма платежа
Отчисления');
 For i := 1 to 10 do
  With Group[i] do
    Writeln (Kod:8, Fam:18, Sum:12:2, Tax:12:2);
 Readln;
End.
```

На данном примере показаны особенности организации доступа к полям записи. В первом фрагменте программы (заполнение полей записи), чтобы получить доступ к полю записи, записывается полное имя соответствующего элемента, например, *Group[i].Fam*. Во втором фрагменте программы (просмотр результата) доступ к полям записи выполняется непосредственно по имени. Данную возможность обеспечивает оператор присоединения, который имеет следующую форму записи:

With <имя записи> do <оператор>;

Контрольные вопросы

1. Дать характеристику типу данных — массив.
2. Как обратиться к элементу (полю) записи.
3. Какие операции допустимы над элементами массива?
4. Дать характеристику строковому типу данных.
5. Как определить количество символов в строке (длину строки)?

6.12.1. Задания

1. Известно расписание поездов, проходящих через станцию: номер поезда, места выезда и прибытия, часы и минуты прибытия, часы и

минуты отправления. Общее число проходящих поездов равно 25. Поезда приходят каждый день. По данному времени определить, какие поезда (номер и назначение) стоят в этот момент на станции. Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по времени отправления.

2. Известна информация о 30 клиентах пункта проката: фамилия, имя, отчество, адрес и домашний телефон. Известно также название предмета, взятого каждым из них напрокат (холодильник, телевизор и т.п.). Вывести на экран фамилию, имя и адрес каждого из клиентов, взявших телевизор. Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии клиента.

3. Дан список группы с результатами сессии (оценки по четырем предметам и 5 зачетов). Считая, что студент, имеющий не более одной тройки, имеет право на стипендию, поставить в соответствие такому студенту в графе "стипендия" знак "+", в противном случае - знак "-". Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии студента.

4. Дан список, состоящий из названия книг, фамилии авторов, названия издания и года издания. Напечатать список книг определенного автора, упорядоченный по годам изданий. Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии автора.

5. Распечатать сумму отчислений по каждому оптовому покупателю с точностью до сотых долей рубля. Отчисления по сумме каждого платежа определяются следующим образом: до 500 рублей - 1.5%, свыше 500 рублей - 3%. Результат напечатать в следующем виде:

Код покупателя	Фамилия	Сумма платежа	Отчисления
		ИТОГО	

Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии оптового покупателя.

6. Дан список выпускников 10 класса и оценки аттестата у каждого. Тем, у кого средний балл аттестата больше или равен 4.5, поставить в графе "эксперимент" знак "+", а остальным знак "-". Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии выпускника.

7. Дан список абитуриентов, средний балл аттестата и оценки на вступительных экзаменах у каждого (по пятибалльной системе). Считая, что проходной балл 22, определить, станет ли данный абитуриент студентом. Если да, то в графе "принят" поставить "да", в противном случае - "нет". Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии абитуриента.

8. Дан список абонентов и номера телефонов. Выделить номера, начинающиеся с 496. Данные представить в виде: фамилия, имя, отчество, адрес, номер телефона. Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии абонента.

9. Дан список группы с оценками экзаменационной сессии. Подсчитать количество отличников в группе и в графе "имеется задолженность" студентам имеющим двойки напечатать "да", а всем остальным "нет". Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии студента.

10. Дан список группы и оценки экзаменационной сессии с названием предметов. Напечатать список, и подсчитать по какому предмету в группе имеется наибольшее количество двоек. Программа должна обеспечивать ввод данных, отображение введенных данных, упорядоченное по фамилии студента.

6.13. Процедуры и функции

Процессу написания программ предшествует, как правило, два этапа (рис. 6.4)

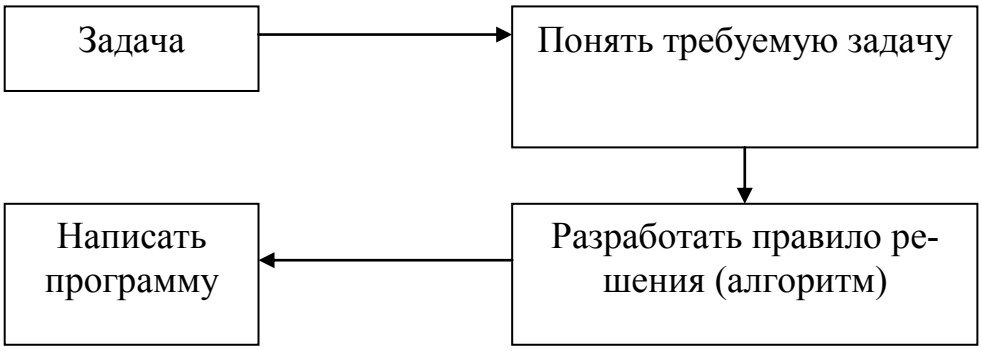


Рис 6.4. Этапы решения задачи

На первом этапе необходимо добиться, чтобы поставленная задача была понята. В случае простой задачи на это может потребоваться лишь несколько секунд. Абсолютно бесполезно приступать к следующему этапу, если то, что должна делать программа осталось непонятным. Только после прояснения этого вопроса можно приступать к следующему этапу — разработке алгоритма, представляющего собой последовательность шагов. Каждый шаг описывает законченное действие.

Из того, что алгоритм представляет собой последовательность шагов, не следует, что при его построении вначале принимается решение о первом шаге, затем о втором, третьем и т.д., пока не будет записан последний шаг.

На самом деле такой метод разработки программ применим только в очень коротких решениях.

Одно из ограничений процесса мышления состоит в том, что в каждый момент времени мы можем в деталях охватить только несколько шагов алгоритма. С другой стороны, если мы хотим убедиться в том, что программа будет работать правильно, то необходимо рассматривать алгоритм как единое целое.

Преодолеть противоречие между необходимостью рассмотреть алгоритм как единое целое и возможностью рассмотреть в каждый момент времени лишь несколько шагов можно с помощью абстрагирования.

Вначале алгоритм записывается в виде последовательности шагов, указывающих, что надо делать, но игнорирующих то, как должно быть сделано это. Далее выполняется детализация каждого шага, то есть каждый шаг расписывается своей последовательностью шагов. В зависимости от сложности задачи и эти более мелкие шаги можно в свою очередь на более мелкие шаги. При этом в каждый момент времени не придется рассматривать большое количество алгоритмических шагов. Размельчение каждого шага будет продолжаться до тех пор, пока для его перевода на алгоритмический язык не будут требоваться дополнительные затраты на разработку. Такой метод конструирования программ получил название «нисходящее программирование». К достоинствам этого метода можно отнести:

- создание хорошо структурированных программ;
- сокращение времени разработки программ;
- увеличение надежности программ;
- упрощение процесса модификации программ.

Базовой управляющей структурой в программировании, позволяющей реализовать нисходящую концепцию разработки программ являются подпрограммы. Именно подпрограммы представляют собой инструмент, с помощью которого любая программа может быть разбита на ряд в известной степени независимых друг от друга частей.

В языке программирования Паскаль определено два вида подпрограмм: процедуры и функции. Процедуры и функции представляют собой во многом самостоятельные фрагменты программы, связанные с основной программой лишь с помощью нескольких параметров. Самостоятельность процедур и функций позволяет локализовать в них все детали

программной реализации того или иного алгоритмического действия и поэтому изменение этих деталей, например, в процессе отладки обычно не приводит к изменениям основной программы.

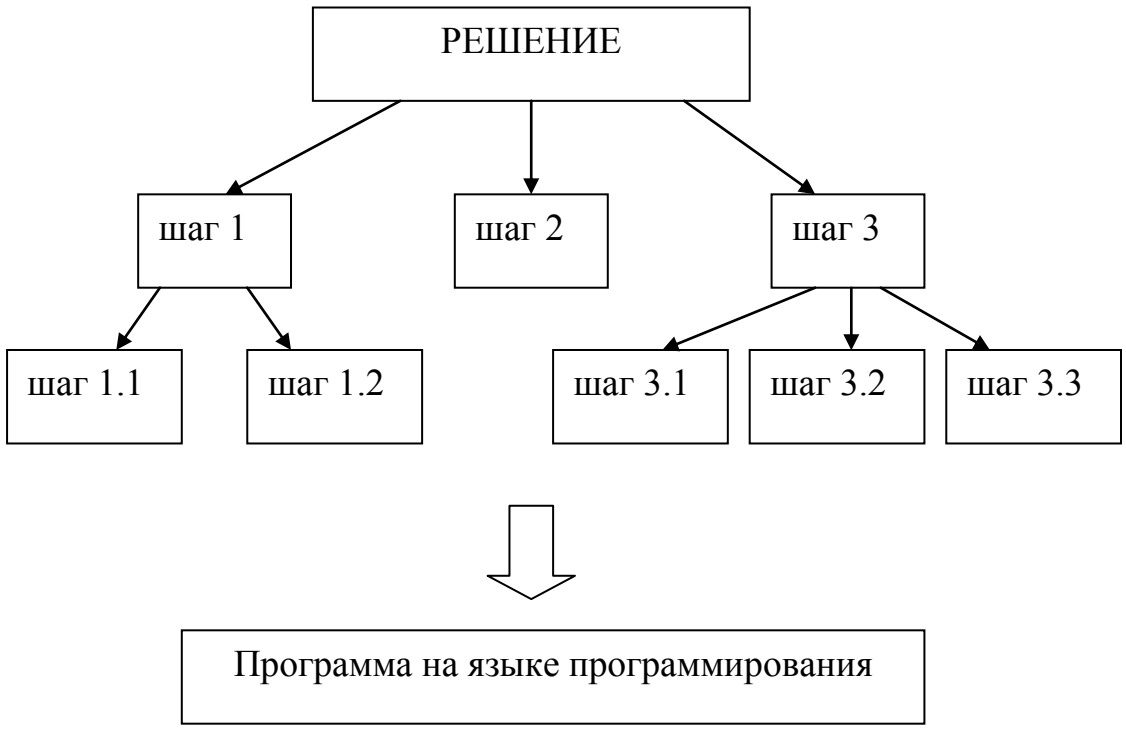


Рис. 3.5 Нисходящая разработка программ

Процедурой в Турбо Паскале называется особым образом оформленный фрагмент программы, имеющий собственное имя. Упоминание этого имени в тексте программы приводит к активизации процедуры и называется ее вызовом. Сразу после активизации процедуры начинают выполняться входящие в нее операторы, после выполнения последнего из них управление возвращается обратно в основную программу, и выполняются операторы, стоящие непосредственно за оператором вызова процедуры (рис.3.6).

Отличие функции от процедуры заключается в том, что результатом исполнения операторов, образующих тело функции, всегда является некоторое единственное значение простого либо строкового типа, по-

этому обращение к функции можно использовать в соответствующих выражениях наряду с переменными и константами.

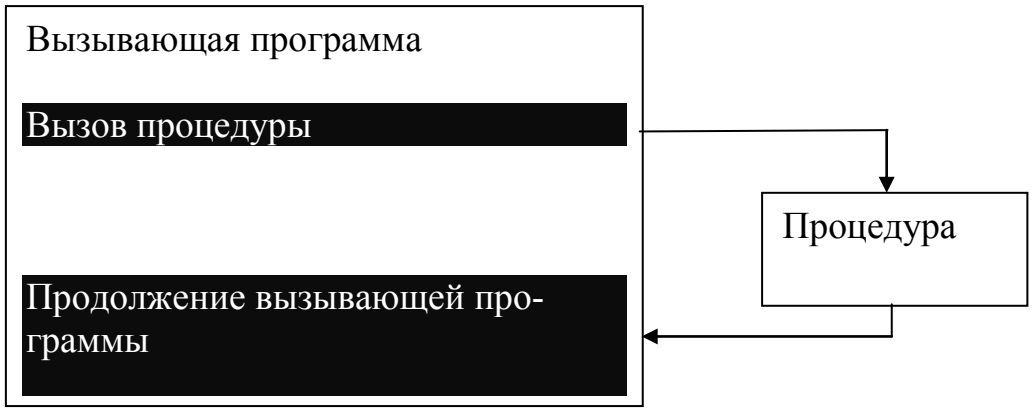


Рис. 3.6 Взаимодействие вызывающей программы и процедуры

Любую процедуру (функцию) в программе необходимо описать в разделе описаний.

Описать подпрограмму — это значит указать ее заголовок и тело. В заголовке объявляются имя подпрограммы и формальные параметры, если они есть. Для функции, кроме того, указывается тип возвращаемого ею результата. За заголовком следует тело подпрограммы, которое, подобно программе, состоит из раздела описаний и раздела исполняемых операторов. В разделе описаний подпрограммы могут встретиться описания подпрограмм низшего уровня, в тех — описания других подпрограмм и т.д.

Рис. 3.7. Пример структуры программы

Структура программы, которая изображена на рис. 3.7. (для простоты считается, что все подпрограммы представляют собой процедуры без параметров) может быть представлена в программе на языке программирования Паскаль следующим описанием:

```
Program Pr;
Procedure A;
Procedure A1;
Begin
{операторы процедуры A1}
End;
Begin
{операторы процедуры A}
End;
Procedure B;
Procedure B1;
Begin
{операторы процедуры B1}
end;
Begin
{Операторы процедуры B}
end;
BEGIN
{ операторы программы Pr}
END.
```

Все имена, используемые в подпрограмме, делятся на локальные и глобальные по отношению к данной подпрограмме. Локальные имена переменных, типов, констант, процедур, функций описываются внутри подпрограммы. Локальные объекты порождаются в процедуре или при каждом входе в подпрограмму и уничтожаются при выходе из этой подпрограммы. Таким образом, со стороны операторов, использующих обращение к подпрограмме, она трактуется как «черный ящик», в котором реализуется тот или иной алгоритм. Все детали этой реализации скрыты от глаз пользователя подпрограммы и потому недоступны ему. Например, в рассмотренном выше примере из основной программы можно обратиться к процедурам *A* и *B,* но нельзя вызвать ни одну из вложенных в них процедур *A1, B1* и т.д.

Из подпрограммы можно ссылаться на любой глобальный объект, описание которого находится выше по отношению к активной на текущий момент подпрограмме. Так, например, из подпрограммы *B1* мы можем вызвать подпрограмму *A,* использовать имена, объявленные в основной программе, в подпрограмме *B* и даже обратиться к ним. Но можно и переопределить любое имя. Если имя какой–либо глобальной переменной переопределено, то в области действий определяющей подпрограммы имеет силу уже новая связь между именем и типом. Глобальная переменная с тем же именем в этой области становится недоступной (если только она не передается как параметр).

6.13.1. Синтаксис описания процедур и функций

Описание подпрограммы состоит из заголовка и тела подпрограммы (раздел описания и раздел операторов).

Описание процедуры имеет вид:

PROCEDURE <имя> [(<сп.ф.п.>)]; {заголовок процедуры}

```
Const
Type
Var                {раздел описания данных процедуры}
Procedure
Function
Begin
{операторы процедуры}
End;
```

Описание функции имеет вид:

```
Function <имя> [(<сп.ф.п.>)]: <тип>;  {заголовок функции}
Const
Type
Var                {раздел описания данных процедуры}
Procedure
Function
Begin
{операторы процедуры}
<имя> := <значение>; {обязательная инструкция}
End;
```

Здесь

<имя> — имя подпрограммы (правильный идентификатор);

<сп.ф.п.> — список формальных параметров (может отсутствовать);

<тип> — тип возвращаемого функцией результата.

6.13.2. Параметры процедур и функций

Необязательный параметр <*сп.ф.п*> — это последовательность разделенных точкой с запятой переменных с указанием их типа, задающих при вызове процедуры передаваемые параметры. Операторы тела подпрограммы рассматривают список формальных параметров как своеобразное расширение раздела описаний: все переменные из этого списка могут использоваться в любых выражениях внутри подпрограммы. Таким способом осуществляется настройка алгоритма подпрограммы на конкретную задачу. Формальные параметры (с их типами), задаваемые при описании процедуры, всегда представляют только имена. В тоже время при вызове процедуры ее аргументы — фактические параметры — могут быть не только переменными, но и выражениями.

Примеры оформления заголовка процедур и функций.

Procedure SB (a : real; b : integer; c : char);
Function ArcSin (x : Real): Real;
Procedure Res (a , b, c : Real);

В заголовке процедуры *Res* параметры описаны через запятую. Такая форма записи допустимы только для однотипных параметров процедуры или функции. Так в процедуре *Res* переменные *a , b, c* — вещественные числа.

Формальные параметры делятся на параметры–значения (входные) и параметры–переменные (выходные). В описании заголовка процедуры параметрам–переменным предшествует ключевое слово *VAR*.

Procedure A1 (k : integer; s: String; Var p : Char);
Здесь *k, s* — параметры–значение, *p* — параметр–переменная.

Параметр–значение используется для передачи данных процедуре или функции. Параметр, представляющий собой результат процедуры, должен определяться как параметр–переменная.

6.13.3. Вызов процедур и функций

Вызов процедуры осуществляется указанием ее имени, за которым следует список фактических параметров: *A1(6, ss, P1);*.

Если список параметров в описании процедуры отсутствует, то вызов процедуры осуществляется только указанием ее имени.

Вызов функции должен входить в выражение:

Z := ArcSin(x);

При вызове процедуры (функции) осуществляется замена формальных параметров на фактические, что позволяет нужным образом настроить алгоритм, реализованный в подпрограмме.

Оформляя инструкцию вызова необходимо соблюдать следующие требования:

1. Количество и типы формальных параметров должно строго соответствовать количеству и типам фактических параметров в момент обращения к подпрограмме.

2. Порядок следования формальных параметров должен соответствовать порядку следования фактических параметров, поскольку смысл используемых фактических параметров зависит от того, в каком порядке они перечислены при вызове подпрограммы.

3. Если формальный параметр объявлен как параметр–переменная, то при вызове подпрограммы ему должен соответствовать фактический параметр в виде переменной нужного типа.

4. Если формальный параметр объявлен как параметр–значение,

то при вызове ему может соответствовать произвольное выражение.

Пример.

В языке Паскаль нет операции возведения в степень, однако с помощью встроенных функций *LN(X)* и *EXP(X)* нетрудно реализовать новую функцию с именем, например, *Power*, осуществляющую возведение любого вещественного числа в любую вещественную степень. Математическое решение данной задачи следующее: $a^b = e^{b \cdot \ln a}$.

```
Var
X, Y  : Real;
Z     : Real;
Function Power(a, b  : Real) : Real;
Begin
Power := Exp(b*Ln(a));
End;
BEGIN
Write ('Введите два вещественных числа: ');
Readln( X, Y);
Writeln ('Значение', X:6,' в степени ',Y:6,' равно ', Power(X, Y):8);
Z := Power(Y, X);
Writeln ('Значение', Y:6, ' в степени ', X:6, ' равно ', Z:8);
End.
```

Функция *Power* при первом вызове просто указана в качестве параметра при обращении к встроенной процедуре *WRITELN*. Второй вызов функции *Power* — выражение правой части оператора присваивания. Параметры *X* и *Y* в момент обращения к функции — это фактические параметры. Они подставляются вместо формальных параметров *A* и *B* в заголовке функции и затем над ними осуществляются нужные действия.

Полученный результат присваивается идентификатору функции — именно он и будет возвращен как значение функции при выходе из нее. В программе функция *Power* вызывается дважды — сначала с параметрами *X* и *Y*, а затем *Y* и *X*, поэтому будут получены два разных результата.

6.13.4. Параметры–массивы

По правилам языка Паскаль типом любого параметра в списке формальных параметров может быть только стандартный или ранее объявленный тип. Поэтому нельзя, например, объявить следующую процедуру:

```
Procedure Sum (A : Array [1..10] of  Real);
```

Инструкция является ошибочной, так как в ней фактически объявляется тип диапазон–диапазон ([1..10]), указывающий границы индексов массива.

Чтобы обойти данное ограничение выполняются следующие описания:

```
Type
Mas = Array [1..10] of Real;
Procedure Sum ( A : Mas);
```

Но вызов данной процедуры вновь имеет ограничение: фактический параметр должен иметь тот же тип *Mas*, что затрудняет использование процедуры для массивов различной размерности. В современных версиях Паскаля данное ограничение снимается следующим описанием заголовка процедуры:

```
Procedure Sum ( A : Array of Real);
```

Пример.

В процедуре *Print_Mas* Реализован алгоритм отображение на экране значений одномерного массива целых чисел.

```
Type Mas = Array[1..6] of integer;
Var
A :  Mas;
B :  Array[1..10] of integer;
i : integer;
Procedure Print_Mas (Var C : Array of Integer; n : byte; S :
string);
{S — строка заголовка}
{n — текущая размерность массива}
{C — массив целых чисел}

Var
i, j : Byte;
Begin
Writeln(S); {вывод на экран заголовочной строки}
For i := 1 to n do Write(C[i]:3);
Writeln;
End;

Begin
{Инициализация массива A случайными числами}
For i := 1 to 6 do A[i] := Random(30);
{ Инициализация массива B случайными числами }
For i := 1 to 10 do B[i] := Random(30);
```

```
Print_Mas (A, 6, 'Элементы массива A');
Print_Mas (B, 10, 'Элементы массива B');
end.
```

6.13.5. Контрольные вопросы.

1. Сформулируйте принцип нисходящего программирования.
2. Что такое подпрограмма?
3. Какие виды подпрограмм определены в Паскале?
4. В чем отличие глобальных и локальных переменных?
5. Что такое параметры подпрограммы?
6. Как оформляются параметры в подпрограммах? Приведите примеры.
7. Как описать выходной параметр?
8. Какие соглашения по параметрам должны выполняться при вызове подпрограмм?
9. Как передать параметром двумерный массив?

6.13.6. Задания.

Для каждого из вариантов требуется составить программу с использованием подпрограммы типа Procedure.

1. Даны три массива A(10), B(12), C(16). Нормировать элементы каждого массива по максимальному.
2. Даны три массива A(9), B(11), C(3). В каждом массиве найти сумму элементов после первого отрицательного и сумму элементов до него.

3. Даны три массива A(11), B(10), C(14). Если первый элемент массива отрицательный, то вычислить сумму элементов на нечетных местах. В противном случае вычислить сумму всех элементов.

4. Даны три массива A(9), B(13), C(15). В каждом из этих массивов элементы с номерами кратными 3 заменить полу суммой двух предыдущих элементов если их произведение меньше 0 и оставить без изменения в противном случае.

5. Даны два массива A(18), B(13). Подсчитав сколько в каждом из них имеется групп рядом расположенных равных между собой элементов.

6. Даны три массива A(11), B(12), C(14). Вычислить произведение элементов после первого отрицательного в каждом массиве, сам этот элемент заменить его модулем.

Для каждого из вариантов требуется составить программу вычисления значения функции с использованием подпрограммы типа function.

1. Даны три квадратные матрицы разного размера. Решить уравнение

$$P \cdot x^2 + D \cdot x + R = 0$$

где P, D, R - минимальные элементы трех массивов.

2. Составить программу вычисления числа сочетаний из n по m по приближенной формуле

$$A_n^m = \exp(\ln(m!)) - \ln(n!) - \ln((n-m)!)$$

3. Дано действительное число y. Получить

$$P = (1.7 \cdot t(0.25) + \frac{2 \cdot t(1+y)}{6 - t(y^2 - 1)},$$

где $t(x)=\sum_{k=1}^{10}\frac{x^{2\cdot k}}{(2\cdot k+1)!}$.

4. Дано действительное число y. Получить

$$P=\frac{0.85+3\cdot t(0.5\cdot y-2)}{t(y+y^2)-7},$$

где $t(x)=\sum_{k=1}^{10}\frac{x^{2\cdot k}}{(2\cdot k)!}$.

5. Даны действительные числа s,t. Получить

$$\frac{f(t,s+t,s\cdot t)+f(5.2,s,s-t)}{f(3,s,t)},$$

где $f(a,b,c)=\frac{3\cdot a+2\cdot b+a\cdot c}{c-|a\cdot b|}$

6. Дано действительное число y. Получить

$$\frac{3.5\cdot(t(0.33)+0.5\cdot t(-y)}{y+t(y-1)^2},$$

где $t(x)=\cos(x)+\cos(x)^2+\cos(x)^3+...+\cos(x)^{30}$.

7. Даны три массива X{x1,...,x12}, Y{y1,...,y10}, C{c1,...,c14} и число Z. Получить

$$T=\begin{cases}\frac{x+y}{2}\text{.....если.....}z<0\\ (x^{2\cdot c}-c)^{\frac{1}{2}}\text{.....если.....}z>0\end{cases}$$

где x, y, c - среднее арифметическое положительных констант соответствующего массива.

ЛИТЕРАТУРА

1. Вьюкова Н.И., Галатенко В.А., Ходулев А.Б. Систематический подход к программированию/Под ред. Ю.М.Баяновского. —М.: Наука. Гл.ред. физ.-мат. лит.,1988. —208с. — (библиотечка программиста)

2. Гуденко Д.А. Сборник задач по программированию. — СПб.:Питер, 2003. — 475 с.:ил.— (Серия КомпАс»)

3. Дал У., Дейкстра Э., Хоор К. Структурное программирование. — М.: Мир, 1975

4. Докукина Т.К. Программирование и алгоритмические языки: Учебник для сред.спец.заведений. — М.: Машиностроение, 1988. — 496с.:ил.

5. Кнут. Искусство программирования для ЭВМ. Т.3. Сортировка и поиск. — М.: Мир, 1978г. — 846с.

6. Культин Н.Б. Turbo Pascal в задачах и примерах. — Спб.:БХВ–Петербург, 2000. —256 с.:ил.

7. Н. Вирт. Алгоритмы + структуры данных = программы. — М.: Мир, 1985 г.

8. Сафьянова Е.Н. Основы Алгоритмизации и программирования. Учебное пособие. — Томск: Томский межвузовский цент дистанционного образования, 2000г. — 111с.

9. Фаронов В.В. Основы Турбо Паскаля. М.: Учебно-инженерный центр «МВТУ–ФЕТО ДИДАКТИК», 1992. — 304 с., ил.

10. Хьюз Дж., Мичтом Дж. Структурный подход к программированию. — М.: Мир, 1980

11. Юркин А. Задачник по программированию. — СПб.:Питер, 2002. —192 с.

12. Boehm C., Jacopini G. Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules // Communications of the ASM. 1966. 9. P.366-371